UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

SISTEMAS INFORMÁTICOS

Complu6IX: Transición del Campus de la Complutense a la tecnología IPv6

Soporte y Simulación

TÍTULO:	Complu6IX: Transición del Campus de la Complutense a la tecnología IPv6 - Soporte y Simulación.
AUTORES:	Sofía Muñóz Vélez, Silvia Corredor Sanz y Alicia Melado Corral
TUTOR:	Rafael Martínez Torres
DEPARTAMENTO:	Sistemas Informáticos y Programación

Agradecimientos:

Este proyecto no hubiera sido posible sin el ánimo y empeño de nuestro tutor de proyecto Rafael Martinez Torres, profesor colaborador del Departamento de Sistemas Informáticos y Programación de la Facultad de Informática, de la Universidad Complutense de Madrid.

También agradecemos el trabajo realizado por los creadores y colaboradores de las herramientas VNUMLGUI y VNUML.

Y por supuesto, al apoyo recibido por nuestras familias.

Gracias.

Índice

PRÓLOGO

¿Por qué Cumplu6IX?

Complu6IX, es el nombre de un proyecto de fin de carrera que está dedicado a la transición del campus de la Universidad Complutense de Madrid a la tecnología IPv6.

Poco después del origen de este proyecto, se vio que era necesario dividirlo en dos partes, una de ellas dedicada al entorno de creación de los escenarios para las simulaciones y del desarrollo teórico de lo que era el proyecto Complu6IX, y otra originada poco después de la creación de Complu6IX como consecuencia de la necesidad de soporte e interfaz gráfico para el proyecto.

En definitiva, el desarrollo de todo este trabajo es este último proyecto que hemos mencionado que complementa al proyecto Complu6IX, dándole soporte mediante la implementación de herramientas de monitorización gráfica y/o modelado visual de redes.

El proyecto Complu6IX utiliza como herramienta de trabajo, VNUML (Virtual Network User Mode Linux), una herramienta basada en software libre cuyo objetivo es la construcción de entornos virtuales o simulaciones, desarrollada dentro del marco del proyecto europeo Euro6IX.

El desarrollo de este proyecto siguió una primera línea de investigación de herramientas de virtualización gráficas, las alternativas existentes, las posibilidades que existían, las ventajas y los inconvenientes. En todo este proceso se asumía el riesgo del desconocimiento completo por parte de los integrantes del grupo en este campo.

Durante esta primera etapa se tuvo constancia de la existencia de un interfaz gráfico de software libre creado para la herramienta VNUML. Desde el primer momento, este software, de nombre VNUMLGUI, se propuso como opción principal de desarrollo, por tanto las líneas de investigación se centraron en él.

Uno de los inconvenientes que surgieron, era la incompatibilidad de versiones. El software VNUMLGUI, obtenido de sourceforge, se encontraba en un estado desfasado y obsoleto con respecto a la versión de VNUML que ya se había empezado a utilizar en el proyecto Complu6IX. Se barajaron varias posibilidades como era el cambio de versión en la utilización de VNUML a una versión anterior compatible con VNUMLGUI. Pero poco después, se descartó esta opción por la pérdida de tiempo, trabajo y recursos que supondría. Por tanto la opción asequible que existía, era la actualización de VNUMLGUI a la versión de la herramienta VNUML que utilizaba el grupo de trabajo Complu6IX.

Posteriormente, durante el proceso de investigación de este software, se dio con un proyecto de fin de carrera de la Universidad Politécnica de Madrid, cuyo objetivo fue la creación de nuevas funcionalidades para la herramienta VNUML, y en cuyo proyecto se habían realizado modificaciones también en el software VNUMLGUI.

Como consecuencia de esto, se descartó el software descargado de sourceforge y se partió de este otro, que se podía obtener del CVS también en sourceforge.

A esta fase inicial de investigación, le siguió la fase de toma de contacto con la tecnología usada en VNUMLGUI para poder continuar con el desarrollo de este software libre. Esto supuso la inmersión en el estudio completo de nuevos lenguajes de programación por parte del equipo de desarrollo.

Feedback.

Durante el periodo de desarrollo de este proyecto, VNUMLGUI fue sometido a un proceso de feedback.

El equipo de desarrollo tomó una postura crítica frente al software, analizando las posibles mejoras sobre él, para aumentar su efectividad, y en definitiva, convertirlo en una herramienta recomendable de utilizar.

El proceso de feedback supuso que los desarrolladores tuvieran que conocer bien el objetivo final de VNUMLGUI: los resultados que se querían lograr, observando cómo podrían ser mejorables, dando su opinión comparándolo con la herramienta VNUML y si realmente facilitaba el uso de ésta.

De esta manera se enfocó el proceso en varias preguntas, una de ellas en qué se veía que podía faltarle a VNUMLGUI para alcanzar su objetivo, y cómo ayudar a hacer que esto fuera posible.

Una de las ventajas de VNUMLGUI es que es una herramienta de software libre y por tanto, siempre estará dispuesta a recibir consejos, perfeccionamiento y reformas para obtener los mejores resultados, de forma que siempre dará por válidas y útiles las opiniones de los desarrolladores. Esto propicia mayor libertad en la expresión de las ideas y mejoras que los programadores proponen para VNUMLGUI.

Durante todo el proceso que duró el desarrollo del proyecto, éste estuvo sometido a feedback, centrándose el equipo de desarrollo en lo que se quería producir, con la posibilidad de ampliar el horizonte de opciones, en caso de dudas o problemas que pudieran surgir.

La primera parte del desarrollo consistió en la actualización de VNUMLGUI para que fuera compatible con la versión de VNUML, mientras que en paralelo se iban estudiando por otra parte herramientas de análisis de protocolos para incluir en este software como posibilidad de mejora. Una vez actualizado el programa, el equipo de Complu6IX que utilizaba VNUML, ya podía simular sus escenarios con ayuda del interfaz gráfico VNUMLGUI.

La siguiente fase del desarrollo fue la introducción de las mejoras en VNUMLGUI que se habían barajado, como la creación de un analizador de protocolos integrado en el propio software y la transformación del programa de síncrono a asíncrono durante el proceso de simulación de los escenarios en VNUMLGUI.

1. INTRODUCCIÓN

Aunque vnuml oculta al usuario la mayoría de los detalles del software de virtualización uml, conviene tener un mínimo de conocimientos sobre estas cuestiones y su funcionamiento.

De modo que antes de conocer en profundidad VNUML, veamos algunos de sus fundamentos:

- Virtualización
- UML

1.1 La virtualización

Como sistema de virtualización que es VNUML, evita el coste de adquirir y gestionar equipos, de ahí su importancia y que sea interesante ahondar más en el conocimiento de esta técnica en términos generales:

La virtualización es una técnica que consiste el alojar una unidad de proceso en un entorno de un equipo anfitrión, el tema en común de todas las tecnologías de virtualización es la de ocultar los detalles técnicos a través de la encapsulación. Esta técnica ha tenido mucho éxito en los últimos años debido a la gran alternativa que suponen respecto a la implementación en sistemas reales, y está siendo usada cada vez. más utilizadas en diversos entornos.

Las principales ventajas de la virtualización son el ahorro de costes de infraestructura y la simplificación de la gestión. No es comparable el gasto que habría de realizarse en una simulación con entorno real que hacerla en un único pc, pese a que este deba ser lo suficientemente potente, debido al abaratamiento en los últimos tiempos de las elementos hardware (ram y disco duro..).

Así mismo un entorno formado por varios equipos interconectados es difícil de gestionar, pero con la virtualización este problema no es tal, ya que sólo debemos actuar sobre un equipo el anfitrión.

Hay dos categorías principales de virtualización:

• **Virtualización de plataforma** que involucra la simulación de máquinas virtuales. Esta es la categoría usada en este proyecto.

• **Virtualización de recursos** que involucra la simulación de recursos combinados, fragmentados o simples.

1.1.1 Virtualización de plataforma

Este tipo de virtualización se basa en una combinación de hardware y software.

La virtualización de plataforma es llevada a cabo en una plataforma de hardware mediante un software "host" (un programa de control) que simula un entorno computacional (máquina virtual) para su software "guest". Este software "guest", que generalmente es un sistema operativo completo, corre como si estuviera instalado en una plataforma de hardware autónoma.

Dependiendo del enfoque dado a la virtualización de plataforma podemos distinguir entre:

a) Emulación

b) Virtualización completa (Full Virtualization),

c) Virtualización parcial

d) Paravirtualización (Paravirtualization).

e) Virtualización a nivel del sistema operativo

f) Virtualización de aplicaciones

a) Emulación

La emulación se basa en crear máquinas virtuales que emulan el hardware de una o varias plataformas hardware distintas. Este tipo de virtualización es la más costosa y la menos eficiente, ya que obliga a simular completamente el comportamiento de la plataforma hardware a emular e implica también que cada instrucción que se ejecute en estas plataformas sea traducida al hardware real.

Sin embargo la emulación tiene características interesantes, como poder ejecutar un sistema operativo diseñado para una plataforma concreta sobre otra plataforma, sin tener que modificarlo, o en el desarrollo de firmware para dispositivos hardware, donde se pueden comenzar estos desarrollos sin tener que esperar a tener disponible el hardware real.

Uno de los ejemplos más destacados de la actualidad es QEMU. QEMU, entre otras cosas, permite emular diferentes plataformas Hardware como x86, x86-64, PowerPC, SPARC o MIPS. Así pues, podríamos tener dentro de un servidor Linux varios equipos x86 o PowerPC, corriendo diferentes versiones de Linux.

b) Virtualización completa

Con este término se denominan aquellas soluciones que permiten ejecutar sistemas operativos huesped (Guest), sin tener que modificarlos, sobre un sistema anfitrión (Host), utilizando en medio un Hypervisor o Virtual Machine Monitor que permite compartir el hardware real. Esta capa intermedia es la encargada de monitorizar los sistemas huesped con el fin de capturar determinadas instrucciones protegidas de acceso al hardware, que no pueden realizar de forma nativa al no tener acceso directo a él.

Su principal ventaja es que los sistemas operativos pueden ejecutarse sin ninguna modificación sobre la plataforma, aunque como inconveniente frente a la emulación, el sistema operativo debe estar soportado en la arquitectura virtualizada.

En lo que respecta al rendimiento, éste es significativamente mayor que en la emulación, pero menor que en una plataforma nativa, debido a la monitorización y la mediación del hypervisor. Sin embargo, recientes incorporaciones técnicas en las plataformas x86 hechas por Intel y AMD, como son Intel VT y AMD-V, han permitido que soluciones basadas en la virtualización completa se acerquen prácticamente al rendimiento nativo.

Hay que tener en cuenta también que la virtualización completa no se refiere a todo el conjunto de hardware disponible en un equipo, sino a sus componentes principales, básicamente el procesador y memoria.

De esta forma, otros periféricos como tarjetas gráficas, de red o de sonido, no se virtualizan. Las máquinas huesped no disponen de los mismos dispositivos que el anfitrión, sino de otros virtuales genéricos. Por ejemplo, si se dispone de una tarjeta nVidia GeForce en el anfitrión, los equipos huesped no verán esta tarjeta sino una genérica Cirrus.

c) Virtualización parcial

La máquina virtual simula múltiples instancias de mucho (pero no de todo) del entorno subyacente del hardware, particularmente address spaces. Este entorno admite compartir recursos y aislar procesos, pero no permite instancias separadas de sistemas operativos "guest".

Aunque no es vista como dentro de la categoría de máquina virtual, históricamente éste fue un importante acercamiento, y fue usado en sistemas como CTSS, el experimental IBM M44/44X, y podría decirse que en sistemas como OS/VS1, OS/VS2 y MVS

d) Paravirtualización

La paravirtualización surgió como una forma de mejorar la eficiencia de las máquinas virtuales y acercarlo al rendimiento nativo. Para ello se basa en que los sistemas virtualizados (huesped) deben estar basados en sistemas operativos especialmente modificados para ejecutarse sobre un Hypervisor. De esta forma no es necesario que éste monitorice todas las instrucciones, sino que los sistemas operativos huesped y anfitrión colaboran en la tarea.

Este es el caso de Xen y **UML** (User mode Linux). En lugar de que VMM tenga que analizar el código, es el código quien invoca al VMM cuando sea necesario. Esta técnica simplifica muchísimo el VMM y ofrece muy buen rendimiento, aunque en el caso concreto de UML el rendimiento es mediocre, se considera dentro de esta categoría ya que UML es una modificación del kernel de Linux para que pueda ejecutarse dentro de otro Linux.

e) Virtualización a nivel del sistema operativo

Virtualizar un servidor físico a nivel del sistema operativo permite a múltiples servidores virtuales aislados y seguros correr en un solo servidor físico. El entorno del sistema operativo "guest" comparte el mismo sistema operativo que el del sistema "host" (el mismo kernel del sistema operativo es usado para implementar el entorno del "guest"). Las aplicaciones que corren en un entorno "guest" dado lo ven como un sistema autónomo. Ejemplos: Linux-VServer, Virtuozzo, OpenVZ, Solaris Containers y FreeBSD Jails.

f) Virtualización de aplicaciones

Consiste en el hecho de correr una desktop o una aplicación de server localmente, usando los recursos locales, en una máquina virtual apropiada. Esto contrasta con correr la aplicación como un software local convencional (software que fueron "instalados" en el sistema). Semejantes aplicaciones virtuales corren en un pequeño entorno virtual que contienen los componentes necesarios para ejecutar, como entradas de registros, archivos, entornos variables, elementos de uso de interfaces y objetos globales. Este entorno virtual actúa como una capa entre la aplicación y el sistema operativo, y elimina los conflictos entre aplicaciones y entre las aplicaciones y el sistema operativo. Los ejemplos incluyen el Java Virtual Machine de Sun, Softricity, Thinstall, Altiris y Trigence (esta metodología de virtualización el claramente diferente a las anteriores; solo una pequeña línea divisoria los separa de entornos de máquinas virtuales como Smalltalk, FORTH, Tel, P-code).

1.1.2 Virtualización de recursos

El concepto básico de la virtualización de plataforma, descripta anteriormente, se extendió a la virtualización de recursos específicos del sistema como la capacidad de almacenamiento, nombre de los espacios y recursos de la red.

En otras palabras, es aquella relacionada a particionar los recursos de un sistema, de forma que cada aplicación obtenga una porción de los mismos. En este caso, el recurso CPU, Memoria y Dispositivos de I/O son compartidos entre diferentes aplicaciones o procesos, los cuales conviven en una misma instancia de sistema operativo, aunque existe la modalidad en la cual a través de zonas virtuales es posible mantener instancias de sistema operativo separados compitiendo por los recursos de un único sistema.

En todos los casos la idea es que cada aplicación obtenga recursos computacionales como si estuviera corriendo en una máquina exclusivamente para esta. El fin último es ofrecer los recursos y evitar la monopolización de los mismos.

1.2 Interfaces gráficos

En términos generales las interfaces gráficas son la manera que los usuarios tienen para comunicarse con el sistema informático de forma amigable, esto es, es un conjunto de elementos hardware y software de una computadora que presentan información al usuario y le permiten interactuar con la información y con el pc. En los últimos años han supuesto una gran revolución en el mundo de la informática, con grandes consecuencias en la industria hardware y software.

Sin las interfaces gráficas el acceso a las computadoras era exclusivo a las personas que tenían un mínimo de conocimiento del entorno de línea de comandos, que era sobre lo que funcionaban las antiguas computadoras, ahí la limitación que suponían para el gran público.

Con las interfaces de usuario el mercado se revoluciona y la informática aparece como alternativa a multitud de procesos aplicable a todos los campos, debido a su facilidad de uso, ello no significa que las interfaces sean ajenas a aquellas personas que tienen unalto nivel de conocimientos, si no todo lo contrario, dan al usuario experto la facilidad de centrarse el objeto de su estudio, facilitando el uso de herramientas adecuadas.

Este es el caso de vnumlgui, que es una herramienta gráfica para usuarios con ciertos conocimientos de informática, redes….que facilita las creaciones de topologías de red concretas (así como los múltiples usos de la herramienta) evitando que el usuario tenga conocimientos de otro aspectos como pueden ser xml, vnuml en los que se basa vnumlgui, pero que el usuario no necesita para su propósito de estudio de redes.

Existen tres puntos de vista distintos en una IU: el del usuario, el del programador y el del diseñador, Cada uno tiene un modelo mental propio de la interfaz, que contiene los conceptos y expectativas acerca de la misma, desarrollados a través de su experiencia

Modelo del usuario:

El usuario tiene su visión personal del sistema, y espera que éste se comporte de una cierta forma. Se puede conocer el modelo del usuario estudiándolo, ya sea realizando tests de usabilidad, entrevistas, o a través de una realimentación. Una interfaz debe facilitar el proceso de crear un modelo efectivo.

Modelo del diseñador:

El diseñador mezcla las necesidades, ideas, deseos del usuario y los materiales de que dispone el programador para diseñar un producto de software. Es un intermediario entre ambos.

El modelo del diseñador describe los objetos que al utiliza el usuario, su presentación mismo y las técnicas de interacción para su manipulación. Consta de tres partes: presentación, interacción y relaciones entre los objetos

-La presentación es lo que primero capta la atención del usuario, pero más tarde pasa a un segundo plano, y adquiere más importancia la interacción con el producto para poder satisfacer sus expectativas. La presentación no es lo más relevante y un abuso en la misma (por ejemplo, en el color) puede ser contraproducente, distrayendo al usuario.

-La segunda parte del modelo define las técnicas de interacción del usuario, a través de diversos dispositivos.

-La tercera es la más importante, y es donde el diseñador determina la metáfora adecuada que encaja con el modelo mental del usuario. El modelo debe comenzar por esta parte e ir hacia arriba. Una vez definidos los objetos del interfaz, los aspectos visuales saldrán de una manera lógica y fácil.

Modelo del programador:

Es el más fácil de visualizar, al poderse especificar formalmente. Está constituido por los objetos que manipula el programador, distintos de los que trata el usuario (ejemplo: el programador llama base de datos a lo que el usuario podría llamar agenda). Estos objetos deben esconderse del usuario.

Los conocimientos del programador incluyen la plataforma de desarrollo, el sistema operativo, las herramientas de desarrollo y especificaciones.

En este trabajo se han intentado aunar las necesidades y conocimientos de los tres modelos, es decir se ha intentado mejorar la herramienta, viendo aquellas cosas que podían mejorarse, desde una visión de usuario, las cosas que podían facilitar el uso al usuario final, que es a quien va dirigida la aplicación. Asimismo se ha tenido en cuenta la visión del diseñador, es decir tener en cuenta las necesidades del usuario, y las limitaciones con las que cuenta el programado para llevarlo a un terreno práctico.

1.3 Analizador de protocolos

1.3.1 Generalidades

¿Qué es un Analizador de protocolos? Un analizador de protocolos es una herramienta que sirve para monitorizar el tráfico existente por la red, y permite analizar las tramas capturadas ya sea en tiempo real o después de haberlas capturado. Por analizar se entiende que el programa puede reconocer a que protocolo pertenece la trama capturada (TCP, ICMP...) y mostrar al usuario la información decodificada en formato legible. De esta forma, el usuario puede ver todo aquello que en un momento concreto está circulando por la red que se está analizando para diferentes finalidades.

Con la ayuda de esta herramienta, un programador puede desarrollar y depurar cualquier programa que transmita y reciba datos en una red, ya que le permite comprobar lo que realmente hace el programa. Además de para los programadores, estos analizadores son muy útiles a todos aquellos que quieran experimentar o comprobar cómo funcionan ciertos protocolos de red.

Pero además de estas funcionalidades didácticas, también sirven para detectar problemas dentro de una red bien sea tráfico, ataques de virus, gusanos, etc. y para analizar el rendimiento (de una o varias redes pudiendo ser remotas), descubrir cuellos de botella, mejorar la velocidad…

Pero debemos distinguir los cortafuegos (o “firewalls”) de los analizadores de tráfico. Mientras un analizador de tráfico sólo puede capturar, mostrar y crear estadísticas de las tramas capturadas, un firewall también es capaz de controlar las comunicaciones según la política de red que tenga definida.

¿Pero qué limitaciones tiene un analizador de protocolos? Mediante Ethernet la red comenzó a ser un medio compartido donde solo se filtraba el tráfico que llegaba a los equipos mediante las tarjetas de Ethernet. Pero cuando los sniffers están en modo promiscuo son capaces de acceder a TODO el tráfico que circula por la red. Desafortunadamente esta capacidad es usada como herranienta de espionaje, y por ello hoy en día se debe prevenir.

Ante esta amenaza un mecanismo de seguridad muy utilizado es añadir una capa intermedia en la capa de aplicación sobre la que realmente va el protocolo que cifre los paquetes y les añade información de control para detectar si alguien los intenta manipular. El protocolo más estándar para implementar esta funcionalidad es TLS (anteriormente conocido como SSL).

Aunque también se puede saltar esta barrera (intentando suplantar la identidad de un nodo, y enviando un ARP), se trata de ataques activos y detectables con herramientas apropiadas como arpwatch (que hace escucha pasiva) o ettercap.

¿Cuando se dice que un analizador de protocolos captura en Modo Promiscuo? Esto se hace cuando se configuran las tarjetas de red en modo promiscuo, así las tarjetas pueden obtener una copia de todas las tramas Ethernet que circulan por el medio de comunicación (cable p.e.) independiente del direccionamiento que estas tramas tengan.

Es decir se deshabilita el filtro que una tarjeta Ethernet utiliza para identificar aquellas tramas cuya Dirección Ethernet coincide con la que tiene asignada, pudiendo capturar las tramas que circulan por el "cable".

Es importante resaltar que hay que tener un gran sentido de ética y responsabilidad del uso de estas herramientas, ya que cómo permiten visualizar los paquetes, y estos pueden ser usados como armas de espionaje.

Existen distintos tipos de analizadores disponibles comercialmente, y de distintos precios. El precio depende, en gran medida, de la capacidad de análisis (el número de protocolos que es capaz de reconocer y decodificar, la calidad con que lo hacen, la interfaz de usuario, y la capacidad gráfica o estadística que posee), de la tecnología de red soportadas (Ethernet, ATM, FDDI...) y de si se trata sólo de software o si es un equipo hardware especializado(los cuáles permiten ver mucho más).

Para poder usar estas herramientas instalados sobre una computadora, es necesario tener una tarjeta de red con capacidad de cambiarse al modo promiscuo (que escucha todo, se puede decir que todas las soporta). En el caso de una red local (LAN), el equipo debe estar conectado hacia un concentrador o hub (y los demás equipos también), o sino, en el caso de un conmutador o switch, debe estar conectado en un puerto que tenga capacidad de PORT MIRRONING.

Los analizadores de protocolos más avanzados y más caros son equipos portátiles de propósito especial que se pueden conectar a cualquier porción física de la red para facilitar el aislamiento de los problemas de transmisión de datos.

Aunque la mayoría de analizadores decodifican los mismos protocolos, algunos funcionan mejor que otros dependiendo del entorno. Veamos algunos de los analizadores más comunes.

1.3.2 TCPDUMP

Este es un programa de Código Abierto de línea de comandos que es común en el mundo UNIX: Linux, Solaris, BSD, Mac OS X, HP-UX y AIX entre otros, donde es necesario tener los privilegios del root para utilizar tcpdump. Es una herramienta sencilla que muestra los paquetes que pasan por una red Local. Hace uso de la librería libpcap para capturar los paquetes que circulan por la red.

Existe una adaptación de tcpdump para los sistemas Windows que se llama WinDump y que hace uso de la librería Winpcap

Está escrito por Van Jacobson, Craig Leres, y Steven McCanne que trabajaban en este tiempo en el Grupo de Investigación de Red del Laboratorio Lawrence Berkeley. Más tarde el programa fue ampliado por Andrew Tridgell.

1.3.3 Wireshark

Wireshark, el nuevo nombre para Ethereal, es una herramienta de código fuente abierto disponible en diferentes plataformas, sobre la mayoría de sistemas operativos Unix y compatibles, incluyendo Linux, Solaris, FreeBSD, NetBSD, OpenBSD, y Mac OS X, así como en Microsoft Windows. Muestra el tráfico que pasa por una red con varias opciones cómo es la decodificación de los paquetes para ver sus opciones y significado, de manera más amigable con opción de crear filtros o estadísticas.

La funcionalidad que provee es similar a la de tcpdump, pero añade una interfaz gráfica y muchas opciones de organización y filtrado de información. Así, permite ver todo el tráfico que pasa a través de una red (usualmente una red Ethernet, aunque es compatible con algunas otras) estableciendo la configuración en modo promiscuo. También incluye una versión basada en texto llamada tshark.

Permite examinar datos de una red viva o de un archivo de captura salvado en disco. Se puede analizar la información capturada, a través de los detalles y sumarios por cada paquete. Wireshark incluye un completo lenguaje para filtrar lo que queremos ver y la habilidad de mostrar el flujo reconstruido de una sesión de TCP.

1.3.4 Agilent Advisor

Agilent ofrece dos productos en el análisis de protocolo, una es la versión de software para PC llamada Advisor Software Edition, y otra es un equipo portátil que trae módulos para utilizarse con diversas interfaces incluyendo STM1, E1, etc. En la versión del software, además de una versión de prueba, los institutos educativos, y Universidades pueden solicitar una exoneración como parte de una herramienta de estudio de redes. Este producto hace funciones muy similares a los otros analizadores.

2. VNUML: SIMULADOR DE REDES

VNUML es una herramienta de virtualización diseñada para crear complejos escenarios de redes, está basada en el software de virtualización UML (user mode linux) que veremos más adelante. Es una potente herramienta que puede ser usada para simular escenarios de red en linux.

Su principal ventaja es que aporta un entorno de simulación real y potente capaz involucrar docenas de nodos evitando el gasto que supondría el traslado a un entorno real.

2.1 User Mode Linux

Como sistema de virtualización VNUML utiliza UML. User-mode Linux (UML) es una modificación del núcleo del sistema operativo Linux para que funcione sobre su propio interfaz de llamadas al sistema. Esto es, permite una ejecución como proceso de usuario encima del núcleo convencional de linux, teniendo asociados sus propios recursos, en otras palabras User Mode Linux es el Kernel de Linux portado a su propia interfaz de llamadas al sistema. Provee una especie de máquina virtual, que corre Linux como un proceso dentro de otra kernel de Linux.

Es una forma segura de correr versiones de Linux y programas. Se pueden correr programas experimentales, probar nuevas kernels o distribuciones y estudiar el interior del kernel, sin arriesgar la instalación principal.

La mejor manera de explicar esto es recordando como actúa el kernel de linux. El kenel ejecuta procesos y se comunica con el hardware. Cuándo un proceso quiere comunicarse con un dispositivo (pantalla, impresora, disco..), le pide al kernel de linux controlar la comunicación con el hardware. Es decir el kernel está en un nivel intermedio entre el hardware y los procesos:

Por su parte, User Mode Linux es un kernel que se va a ejecutar como un proceso, esto es lo que queríamos decir con lo de: 'es el Kernel de Linux portado a su propia interfaz de llamadas al sistema'. La diferencia entre el kernel de UML y uno ordinario, es que el del UML nos se comunica directamente con el hardware, si no que esta comunicación la establece a través del kernel del sistema anfitrión. Esta comunicación se hace de forma muy sencilla, es decir, no requiere muchas cabeceras de traducción ya que ambos (el sistema virtual y el host) son linux.

Inicialmente UML se creó para que los desarrolladores del núcleo pudieran probar versiones inestables del núcleo sobre un sistema en funcionamiento. Como el núcleo en prueba es sólo un proceso de usuario, si se cuelga, no compromete al sistema que lo aloja.

Entre las ventajas que aporta UML se encuentran la siguientes:

1. Si UML se bloquea, el kernel del huésped seguirá sin problemas.

2. Se puede correr una kernel de Linux sin ser usar privilegios de root.

3. Se puede depurar el UML como cualquier proceso normal.

4. Se puede jugar con el Kernel sin dañar cosas.

5. Se puede usar UML para proveer hosting virtual.

6. Se puede usar como un sandbox para probar aplicaciones nuevas.

7. Se puede usar para probar kernels en desarrollo de manera segura.

8. Se pueden correr diferentes distribuciones al tiempo.

Pero además, el uso de UML permite muchas posibilidades:

- Creación de honeypots son sistemas para para probar la seguridad de una máquina sin comprometerla, es decir comprueba las amenazas de seguridad que van contra ella, no es una red de producción; VNUML facilita la implantación y gestión de este tipo de redes.

- Ejecución de servicios de red. Al ejecutar servicios de red en diferentes procesos UML de una misma máquina se los aisla unos de otros, de forma que no pueden comprometer mutuamente la estabilidad o seguridad de los demás.

- Realizar pruebas con software inestable o incompatible con la versión del núcleo del sistema que aloja el UML (las versiones del núcleo de ambos sistemas puedes ser distintas).

- Permite tener acceso (aparentemente) de administrador a una máquina (principalmente interesante en para servidores virtuales o para entornos educativos en los que se limita las capacidades de un root).

UML no se limita a ser una máquina de virtualización, si no que además proporciona los mecanismos de red para conectar la máquina virtual con el sistema host o con otros sitemas virtuales, la idea principal en la interconexión de redes con UML es que nos facilitan distintas opciones en la capa de transporte para la administración de paquetes entre el sistema virtual y su host.

A continuación mostramos algunos de los tipos de transporte disponibles en UML:

- Etherap, TUN/TAP: Transportes usados para el intercambio de paquetes entre el sistema virtual y el real.

- Switch daemon: Un transporte diseñado para la interconexión de redes virtuales con otros sistemas UML.

- Slip, slirp: Transporte usado principalmente cuando Etherap o TUN/TAP no están disponibles o si no tenemos acceso de root a la configuración de redes del host.

- Pcap: Un transporte que facilita una interfaz de red de solo lectura, con lo que es una buena opción para la monitorización de redes.

Pero hay que tener en cuenta que la tarjeta de red emulada en el UML, no tiene conexión directa a la red, por lo que habrá que emular también su conexión para que se pueda disponer de los servicios de red. Para habilitar el dispositivo de red en la máquina virtual hay que pasarle al kernel por línea de comandos una sentencia como esta:

eth<n>=<transport>, <transport args>, donde <n> representa la interfaz del host real, al que se conectará la máquina virtual.

Por ejemplo un dispositivo ethernet (eth0) de la máquina virtual puede ser conectado a un dispositivo ethertap del host:

eth0=ethertap, tap0, fe:fd:0:0:0:1, 192.168.0.254

La explicación teórica de esto es que en la máquina UML hay un dispositivo eth0 que corresponde al dispositivo tap0 del host real. Esta interfaz tap0 se conecta directamente a eth0 del host real.

Aquí es donde entra en juego las interfaces TUN/TAP. Podemos definirlos como un dispositivo virtual punto a punto, de ello hablaremos más adelante.

El uso combinado de VNUML/UML tiene un importante consumo de recursos cobre el equipo anfitrión que debe ser tenido en cuenta (CPU, memoria de disco y memoria física).

Ahora que ya tenemos un conocimiento mayor de todos los aspectos implicados, podemos centrarnos en definitiva el tema a tratar en este capítulo VNUML.

2.2 Virtual Network UML

Es una herramienta basada en software libre, que tiene como objetivo la simulación de redes virtuales basadas en el software de virtualización UML.

Inicialmente desarrollada en el contexto de Euro6IX, un proyecto de investigación para simular escenario IPv6 sobre linux.

Como ya hemos comentado anteriormente, su principal objetivo es abstraer la simulación a un nivel descriptivo, ocultando al usuario los detalles técnicos en los que se basa.

Está compuesta fundamentalmente por dos componentes:

La parte relativa a la especificación de las simulaciones, realizada en lenguaje XML y el intérprete del lenguaje, esto es el parser (vnumlparser.pl) realizado en lenguaje perl, que es el encargado de hacer la simulación transparente al usuario.

XML

XML, sigla en inglés de eXtensible Markup Language («lenguaje de marcas extensible»), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C).Es el lenguaje base para describir entornos de simulación

Pese a ser XML un tecnología sencilla, tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores, de ahí su elección para esta herramienta pese a que hay lenguajes de etiquetas más eficientes. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

DTD

Siglas de Document Type Definition. La definición de tipo de documento (DTD) es una descripción de estructura y sintaxis de un documento XML o SGML. Su función básica es la descripción del formato de datos, para usar un formato común y mantener la consistencia entre todos los documentos que utilicen la misma DTD, es como la gramática.

La DTD especifica restricciones en la estructura y sintaxis del mismo. Se puede incluir dentro del archivo del documento, pero normalmente se almacena en un fichero ASCII de texto separado. La definición de una DTD especifica la sintaxis de una aplicación de SGML o XML, que puede ser un estándar ampliamente utilizado como XHTML o una aplicación local.

La forma en que trabaja el programa se divide en dos etapas:

La primera tiene como objetivo la construcción, en la memoria del programa, de un árbol, llamado árbol DOM (Dinamic Object Model), que contenga toda la información xml. Este árbol se realiza a partir del leguaje fuente de VNUML mediante un parseador de XML basado en DTD (Document Type Definition).Se utiliza DOM ya que la lógica de control necesita acceder a los datos de forma arbitraria y no secuencial.

La segunda parte es la invocación de ciertos comandos (para arrancar, parar, o gestionar la simulación) en el equipo anfitrión a partir del árbol DOM ya creado. Esta es la parte que correspondería realizar al usuario sin la existencia de VNUML.

Esto es no existe una lógica de control como tal, si no que el parser utiliza cuatro modos de funcionamiento:

- Construcción de la topología
- Parada de la simulación
- Arranque de la simulación
- Destrucción de la topología

El parser está íntegramente escrito en perl, se eligió este lenguaje ya que está especialmente diseñado para el procesamiento de expresiones de texto y scripts para servidores web, además de por su modularidad y orientación a objetos.

2.2.1 Modo de funcionamiento

2.2.2 VNUMLparser

El paquete VNUML tiene dos componentes obligatorios.

Por una parte, un lenguaje de especificación de escenarios de red, basado en XML, con su propia sintaxis (especificada dentro del propio modelo del lenguaje a través de validación por DTD) y semántica (especificada en la documentación que acompaña a la herramienta). Se trata de un lenguaje de alto nivel, centrado en conceptos de redes como son nodo, enlace, interfaz de red, etc., y cuyo objetivo es aislar al usuario de detalles complejos sobre UML o virtualización.

El segundo componente es la herramienta VNUML propiamente dicha, que consiste en un parser (intérprete) del lenguaje anteriormente descrito. El programa interpreta el escenario descrito en XML e interactúa con el sistema anfitrión y las máquinas virtuales para crear escenarios, eliminarlos o ejecutar secuencias programas de comandos en los mismos.

VNUMLGUI, es una aplicación que ayuda a crear, editar y simular topologías con la herramienta VNUML sin necesidad de conocer la sintaxis de los ficheros XML definida por la DTD, por lo tanto desarrolla e integra las 2 partes anteriores de un modo sencillo e intuitivo.

Vnumlparser es una herramienta que forma parte del paquete VNUML. Es el intérprete del lenguaje que construye y maneja la simulación.

Para comprobar si una especificación VNUML es correcta se analiza el correspondiente archivo XML con el módulo "vnumlparser.pl". Éste es un fichero escrito en lenguaje Perl cuya función es comprobar que la especificación escrita en el archivo XML es correcta, si no lo es lo indica con mensajes de error.

Cuando se crea una especificación VNUML o se abre el fichero correspondiente de una especificación con VNUMLGUI, éste posee en la ventana principal un componente botón cuyo evento es la llamada al analizador vnumlparser.pl para comprobar que el archivo generado es correcto.

2.3 Un editor gráfico: VNUMLGUI

Ahora que conocemos una poco más los fundamentos teóricos en los que se basa vnumlgui, vamos a hacer una pequeña introducción a esta herramienta, su utilidad y su funcionamiento.

Vnumlgui es una herramienta gráfica que facilita crear, editar y simular simulaciones vnuml, ya que evita la creación de archivos xml, esto es, como su nombre bien indica es una GUI para vnuml.

Además ofrece un soporte gráfico que facilita su utilización, de modo que permite crear y redes, mediante la interconexión de routers (VMs) y switches (Nets), de modo que permite crear la topología necesaria para poder crear el entorno que el usuario desee.

2.3.1 Estados del simulador

Vnumlgui seguirá un modelo orientado a objetos, en el que cada elemento de la simulación es representado como una clase. Además, sigue el modelo de máquinas de estado, en el que se indica los diferentes estados posibles, y cuáles son los estados que pueden seguir a cada uno de ellos. Los estados en los que se puede encontrar la simulación son:

• Void, estado inicial, no hay representado ningún escenario en la herramienta.

• Clean, simulación que se ha salvado y no se ha modificado desde entonces.

• Dirty, simulación con modificaciones desde la última vez que se salvó.

• Building, simulación en espera de que vnumlparser construya la topología.

• Running, simulación cuya topología ya ha sido construida por vnumlparser.

• Execing, simulación ejecutando los comandos que se han especificado.

• Releasing, simulación que el usuario ha mandado finalizar.

El definir los estados en los que es posible se encuentre la simulación es muy útil para la interfaz gráfica. Por ejemplo, permite en caso de estar en el estado de running, se dibuje cada elemento de la simulación con otro color al utilizado en el resto de los estados, para que el usuario sepa a primera vista si las máquinas virtuales están en ejecución o no.

2.3.2 Tecnologías de la simulación

2.3.2.1 Resumen de la tecnología de la simulación

Debido a que nuestro trabajo parte de un proyecto inicial, VNUMLGUI, en el que contribuimos parcialmente a su desarrollo, la parte de elección de la tecnología está ya resuelta.

El desarrollo del proyecto conlleva el conocimiento de tres tecnologías principales: XML, lenguaje Perl y librerías GTK.

Los escenarios de simulaciones de VNUML se describen mediante XML, estos escenarios se pueden crear a mano y abrirlos posteriormente en VNUMLGUI para visualizarlos, o directamente se pueden crear las especificaciones de las simulaciones mediante VNUMLGUI y guardarlas en un documento XML.

La tecnología XML permite la integración de aplicaciones y la posibilidad de compartir, ofrecer y distribuir información en entornos de red. Se permite integrar los datos, visualizarlos, manipularlos, guardarlos y exportarlos con independencia de cuál es su origen, y por tanto la información puede ser utilizada por VNUML y VNUMLGUI con posibilidad de intercambio de datos de manera sencilla y fácil.

Una de las mayores utilidades de XML es poder etiquetar los datos con su significado. Permite la reestructuración de la información, ello conlleva el tratamiento de la información a límites insospechados.

Por otra parte, todo el código fuente de VNUMLGUI está implementado en el lenguaje de programación Perl.

Perl es un lenguaje de propósito general, interpretado, orientado a objeto, y enfocado al tratamiento de información y su presentación por pantalla u otro medio.

La decisión del uso de este lenguaje para el desarrollo de VNUMLGUI, radica en los beneficios que ofrece frente a otros lenguajes. Perl es ideal para desarrollo rápido de aplicaciones, aunque su modularidad permite que se desarrollen también aplicaciones más complejas.

Una de las grandes ventajas del Perl es la comunidad de usuarios: aparte de los grupos de Perl Mongers (usuarios de Perl), hay listas de correo, grupos de noticias, y foros en la Web, donde plantear y resolver las dudas que surjan en el desarrollo de una aplicación.

Veamos algunas de las ventajas de este lenguaje y que posiblemente sean las causantes de su elección para el desarrollo de VNUMLGUI:

- Se pueden juntar varios programas de una forma sencilla para alcanzar una meta determinada.

- Es relativamente rápido para un lenguaje tipo "script".

- Está disponible en múltiples plataformas y sistemas operativos. De hecho funciona bajo diferentes sabores de UNIX, Linux y todo tipo de Windows. Un programa que se implemente teniendo en cuenta la compatibilidad, puede ser escrito en una plataforma y ejecutado en otra.

- Hay una colección muy variada de módulos que pueden ser incorporados a cualquier "script" de Perl. Están disponibles en el CPAN ("Comprehensive Perl Archive Network"), y ello fomenta el uso de este lenguaje.

- Perl es gratuito. Mucho más que eso, es "Software Libre". Esto quiere decir que el código fuente está disponible para que cualquiera lo pueda ver o modificar, y lo que es más importante, siempre lo estará. Los desarrolladores que usan Perl, preservan este objetivo de proporcionar disponibilidad del código para que futuros sucesores continúen y contribuyan al desarrollo de los programas.

- Le otorga al programador mucha libertad para que haga el programa como prefiera. Tal como dice el eslogan de Perl "Hay más de una forma de hacerlo".

- El mantenimiento y depuración de un programa en PERL es mucho más sencillo que la de cualquier programa en C.

Para el desarrollo del interfaz gráfico de VNUMGUI se optó por el módulo Perl-GTK.

GTK significa Gimp ToolKit, es una biblioteca creada por los desarrolladores del programa para manipulación de gráficos The Gimp, que luego se extendió a la mayoría de los programas en GNU/Linux y también en Windows. La biblioteca GTK original fue hecha en lenguaje C. La flexibilidad de Perl permite adaptar todo este código (que está disponible por ser Software Libre) a Perl, de una manera muy sencilla, ya que no hay que reescribir todo, sino simplemente crear un conector que una la biblioteca GTK con el código Perl. GTK es una biblioteca creada para hacer más sencillo el desarrollo de aplicaciones gráficas. Está orientada a eventos, es decir que el usuario genera una serie de eventos (click, enter, mover el mouse, etc), y el programa lleva a cabo distintas acciones según los eventos que se van sucediendo.

Es sencillo apreciar la facilidad con la que Perl nos permite usar GTK de una manera Orientada a Objetos (a la manera de verlos en Perl) y hacer uso de una forma agradable la llamada de los métodos correspondientes a la clase. Este tipo de libertades son muy propias de Perl.

2.3.2.2 Tecnología XML

El Lenguaje Extensible de Marcas, abreviado XML, describe una clase de objetos de datos llamados documentos XML y define parcialmente el comportamiento de los programas de computadora que los procesan. Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

XML tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

Los documentos XML están compuestos por unidades de almacenamiento llamadas entidades, que contienen tanto datos analizados como no analizados.

Los datos analizados están compuestos de caracteres, algunos de los cuales, de la forma datos carácter, y otros de la forma marca.

Las marcas codifican una descripción de la estructura de almacenamiento del documento y su estructura lógica. XML proporciona un mecanismo para imponer restricciones al almacenamiento y a la estructura lógica.

Se utiliza un módulo software llamado procesador XML para leer documentos XML y proporcionar acceso a su contenido y estructura. Se asume que un procesador XML hace su trabajo dentro de otro módulo, llamado aplicación. Esta especificación describe el comportamiento requerido de un procesador XML en términos de cómo leer datos XML y la información que debe proporcionar a la aplicación.

Origen y Objetivos:

XML fue desarrollado por un Grupo de Trabajo XML (originalmente conocido como "SGML Editorial Review Board") formado bajo los auspicios del Consorcio World Wide Web (W3C), en 1996. Fue presidido por Jon Bosak de Sun Microsystems con la participación activa de un Grupo Especial de Interés en XML (previamente conocido como Grupo de Trabajo SGML) también organizado en el W3C.

Los objetivos de diseño de XML son:

- XML debe ser directamente utilizable sobre Internet.
- XML debe soportar una amplia variedad de aplicaciones.
- XML debe ser compatible con SGML.
- Debe ser fácil la escritura de programas que procesen documentos XML.
- El número de características opcionales en XML debe ser absolutamente mínima, idealmente cero.
- Los documentos XML deben ser legibles por humanos y razonablemente claros.

- El diseño de XML debe ser preparado rápidamente.
- El diseño de XML debe ser formal y conciso.
- Los documentos XML deben ser fácilmente creables.
- La concisión en las marcas XML es de mínima importancia.

El uso de tecnologías XML, combinado con Tecnologías Web, ofrece una serie de ventajas como son:

- Estandarización.
- Integración de aplicaciones.
- Complementario al uso de Tecnologías Web.
- Portabilidad de la información.
- Accesibilidad a la información.

Documentos XML:

Un objeto de datos es un documento XML si esta bien-formado. Un documento XML bien-formado puede además ser válido si cumple una serie de restricciones.

Éste es uno de los aspectos más importantes de este lenguaje, así que hace falta entender bien la diferencia entre documentos bien formados y documentos válidos:

- Bien formados: son todos los que cumplen las especificaciones del lenguaje respecto a las reglas sintácticas, sin estar sujetos a unos elementos fijados en un DTD. De hecho los documentos XML deben tener una estructura jerárquica muy estricta, y los documentos bien formados deben cumplirla.
- Válidos: Además de estar bien formados, siguen una estructura y una semántica determinada por un DTD: sus elementos y sobre todo la estructura jerárquica que define el DTD, además de los atributos, deben ajustarse a lo que el DTD dicte.

Todo documento XML posee una estructura lógica y una física. Físicamente, el documento está compuesto de unidades llamadas entidades. Una entidad puede hacer referencia a otras entidades para que éstas sean incluidas en el documento.

Un documento comienza en una "raíz" o entidad documento. Lógicamente, el documento está compuesto de declaraciones, elementos, comentarios, referencias a caracteres e instrucciones de procesamiento, todos los cuales se indican en el documento mediante marcas explícitas.

Declaración de Tipo de Documento:

La declaración de tipo de documento XML, contiene o describe declaraciones de marcas que proporcionan una gramática para una clase de documentos. Esta gramática se conoce como definición de tipo de documento o DTD.

La declaración de tipo de documento puede referirse a un subconjunto externo (un tipo especial de entidad externa) que contiene declaraciones, o puede contener las declaraciones de marcas directamente en un subconjunto interno, o puede hacer ambos. Una DTD para un documento consiste en la unión de ambos subconjuntos.

Cada documento pertenece a un vocabulario fijo, establecido por la DTD. No se pueden combinar elementos de diferentes vocabularios. Asimismo es imposible para un intérprete (por ejemplo un navegador) analizar el documento sin tener conocimiento de su gramática (del DTD).

La DTD define los tipos de elementos, atributos y entidades permitidas, y puede expresar algunas limitaciones para combinarlos. Como ya se ha explicado anteriormente, los documentos XML que se ajustan a su DTD se denominan válidos.

Documentos XML bien formados:

Se llama documentos "bien formados" a los documentos que cumplen con todas las definiciones básicas de formato y pueden, por lo tanto, ser analizados correctamente por cualquier "parser" (Analizador Sintáctico) que cumpla con la norma.

Los documentos han de seguir una estructura estrictamente jerárquica con lo que respecta a las etiquetas que delimitan sus elementos. Una etiqueta debe estar correctamente incluida en otra, es decir, las etiquetas deben estar correctamente anidadas. Los elementos con contenido deben estar correctamente cerrados.

Los documentos XML sólo permiten un elemento raíz del que todos los demás sean parte, es decir, sólo puede tener un elemento inicial.

Los valores atributos en XML siempre deben estar encerrados entre comillas simples o dobles.

El XML es sensible a mayúsculas y minúsculas. Existe un conjunto de caracteres llamados espacios en blanco (espacios, tabuladores, retornos de carro, saltos de línea) que los procesadores XML tratan de forma diferente en el marcado XML.

Es necesario asignar nombres a las estructuras, tipos de elementos, entidades, elementos particulares, etc. En XML los nombres tienen alguna característica en común.

Las construcciones como etiquetas, referencias de entidad y declaraciones se denominan marcas; son partes del documento que el procesador XML espera entender. El resto del documento entre marcas son los datos entendibles por las personas.

Partes de un documento XML.

Un documento XML está formado por el prólogo y por el cuerpo del documento.

Prólogo:

Aunque no es obligatorio, los documentos XML pueden empezar con unas líneas que describen la versión XML, el tipo de documento y otras cosas.

El prólogo contiene:

- Una declaración XML. Es la sentencia que declara al documento como un documento XML.
- Una declaración de tipo de documento. Enlaza el documento con su DTD, o el DTD puede estar incluido en la propia declaración o ambas cosas al mismo tiempo.
- Uno o más comentarios e instrucciones de procesamiento.

Cuerpo:

A diferencia del prólogo, el cuerpo no es opcional en un documento XML, el cuerpo debe contener al menos un elemento raíz, característica indispensable también para que el documento esté bien formado.

Elementos.

Los elementos XML pueden tener contenido (más elementos, caracteres o ambos), o bien ser elementos vacíos.

Atributos.

Los elementos pueden tener atributos, que son una manera de incorporar características o propiedades a los elementos de un documento.

Secciones CDATA.

Permiten integrar texto en un documento en XML que de otra forma sería interpretado como etiquetas. Es decir, estamos introduciendo texto que luego el procesador XML va a mostrar pero no va a procesar como marcado.

Los CDATA empiezan con los caracteres "<![CDATA[" y termina con "]]>".

Dentro de ellos podemos colocar cualquier cosa ya que no va a ser interpretado, con la salvedad de la cadena que indica el final de CDATA, "]]>", ya que el procesador al encontrársela entendería que la sección CDATA ya ha terminado con las nefastas consecuencias que ésto puede tener.

Entidades predefinidas.

En XML existen algunos caracteres reservados que no podemos utilizar para evitar problemas con el marcado, lo que no significa que no tengan que salir en nuestros documentos XML.

Por ejemplo, nos aparece el caso cuando intentamos escribir la etiqueta <console>: Esta parte describe la etiqueta <console> .

Una posible solución es la utilización de CDATA, pero sin duda es poco útil cuando simplemente queremos escribir un carácter.

Las entidades predefinidas son marcas XML que se utilizan para representar estos caracteres. El XML especifica cinco entidades predefinidas: & para el &, < para el <, > para el >, ' para el ', " para el ",

Como podemos observar, se reconocen al ir entre los símbolos "&" y ";".

Comentarios.

Mediante los cuales podemos proporcionar información que el parser no tendrá en cuenta. Los comentarios empiezan con los caracteres "<!--" y terminan con "-->" y pueden colocarse en cualquier sitio excepto dentro de las declaraciones, etiquetas y otros comentarios.

2.3.2.3 Perl: Un lenguaje de Scriptin

Perl significa Practical Extraction and Report Language, algo así como lenguaje práctico de extracción y de informes. Es un lenguaje de programación diseñado por Larry Wall creado en 1987. Perl toma características del C, del lenguaje interpretado shell (sh), AWK, sed, Lisp y, en un grado inferior, muchos otros lenguajes de programación.

Estructuralmente, Perl está basado en un estilo de bloques como los del C o AWK, y fue ampliamente adoptado por su destreza en el procesado de texto y no tener ninguna de las limitaciones de los otros lenguajes de script.

Perl es un lenguaje interpretado, aunque en realidad, el intérprete de Perl, como todos los intérpretes modernos, compila los programas antes de ejecutarlos. Por eso se habla de scripts, y no de programas, concepto referido principalmente a programas compilados al lenguaje máquina nativo del ordenador y sistema operativo en el que se ejecuta.

La estructura completa de Perl deriva ampliamente del lenguaje C. Perl es un lenguaje imperativo, con variables, expresiones, asignaciones, bloques de código (delimitados por llaves), estructuras de control y subrutinas.

La primera versión de PERL que llegó a ser suficientemente conocida fue la versión 4. Esta versión se estuvo desarrollando desde 1991 a 1993, y coincidió con la popularidad del PERL como lenguaje para programación de servidores de Internet; aunque originalmente se había diseñado como lenguaje para administración de sistemas. La versión 5 estable no apareció hasta octubre de 1994, y ha sido tan popular que todavía se usa. Introdujo muchas de las características que hacen al PERL un lenguaje tan flexible, incluyendo los módulos, las facilidades para programación dirigida a objetos, referencias y mucha mejor documentación. Aparecen muchos otros libros, tales como Learning Perl.

Aplicaciones del lenguaje Perl.

El lenguaje Perl prácticamente, sirve para todo. Todas las tareas de administración de UNIX se pueden simplificar con un programa en Perl. Se usa también para tratamiento y generación de ficheros de texto. También hay proyectos completos y complejos escritos en Perl, pero son los menos.

Últimamente ha encontrado su aplicación en la escritura de CGI (Common Gateway Interface), o scripts ejecutados desde páginas de la World Wide Web. Grandes proyectos escritos en Perl son Slash, IMDb y UseModWiki, un motor de Wiki. Muchos sitios web con alto tráfico, como Amazon.com y Ticketmaster.com usan Perl extensamente.

Se dice de Perl que es un lenguaje de propósito general originalmente desarrollado para la manipulación de texto y que ahora es utilizado para un amplio rango de tareas incluyendo administración de sistemas, desarrollo web, programación en red, desarrollo de GUI y más.

Se previó que fuera práctico (simple, eficiente, completo) en lugar de hermoso (pequeño, elegante, mínimo). Sus principales características son que no es un lenguaje excesivamente complejo, soporta tanto la programación estructurada como la programación orientada a objetos y la programación funcional, tiene incorporado un poderoso sistema de procesamiento de texto y una enorme colección de módulos disponibles.

Implementación.

Perl está implementado como un intérprete, escrito en C, junto con una gran colección de módulos, escritos en Perl y C.

El intérprete tiene una arquitectura orientada a objetos. Todos los elementos del lenguaje Perl — escalares, listas, hashes, referencias a código, manejadores de archivo — están representados en el intérprete como estructuras C. Las operaciones sobre estas estructuras están definidas como una numerosa colección de macros, typedef y funciones; esto constituye la API C de Perl.

La ejecución de un programa Perl se puede dividir en dos fases: tiempo de compilación y tiempo de ejecución. En tiempo de compilación el intérprete parsea el texto del programa en un árbol sintáctico. En tiempo de ejecución, ejecuta el programa siguiendo el árbol. El texto es parseado sólo una vez y el árbol sintáctico es optimizado antes de ser ejecutado, para que la fase de ejecución sea relativamente eficiente.

Estructura del lenguaje.

Vamos a recurrir a ejemplos para explicar este apartado.

En Perl, el programa "Hola mundo" es:

```
#!/usr/bin/perl –w

use strict;

print "¡Hola mundo!\n";
```

La primera línea contiene el shebang (par de caracteres que identifica el texto que sigue), que le indica al sistema operativo dónde encontrar el intérprete de Perl. La segunda línea introduce el pragma strict que se usa en grandes proyectos de software para el control de calidad. La tercera imprime el string ¡Hola mundo! y un carácter de nueva línea.

Perl tiene tres tipos de datos: escalares, listas y hashes:

- Un escalar es un solo valor; puede ser un número, un string (cadena de caracteres) o una referencia.

- Un lista es una colección ordenada de escalares (una variable que almacena una lista se llama array).

- Un hash, o memoria asociativa, es un mapeo de strings a escalares; los strings se llaman claves y los escalares valores.

$var # un escalar

@var # un array

%var # un hash

Perl también tiene un contexto booleano que utiliza en la evaluación de declaraciones condicionales. Los siguientes valores en Perl evalúan todos como falso:

$falso = 0; # el número cero

$falso = 0.0; # el número cero como flotante

$falso = '0'; # el string cero

$falso = ""; # el string vacío

$falso = undef; # el valor devuelto por undef

Todos los demás valores se evalúan a verdadero. Esto incluye el curioso string auto-descriptivo "0 pero verdadero", que de hecho es 0 como número, pero verdadero como booleano.

La función defined() le dice si la variable tiene algún valor. En el ejemplo anterior defined($falso) será verdadero con cada uno de los valores anteriores, excepto undef.

Una lista se define listando sus elementos, separados por comas y rodeados por paréntesis donde así sea requerido por la precedencia de los operadores.

@numeros = (20, 10, 30, 15);

Un hash puede ser inicializado desde una lista de pares clave/valor.

%consoles = (console => 'xterm', id => '1');

Ejemplo de Acceso a los elementos:

$numero[2] # un elemento de @numeros

$consoles{id} # un valor de %consoles

Estructuras de control.

Perl tiene varias clases de estructuras de control. Las condiciones están rodeadas por paréntesis y los bloques subordinados por llaves:

etiqueta while (condición) {... }

etiqueta while (condición) {... } continue {... }

etiqueta for (expresión inicial; expresión condicional; expresión incremental) {... }

etiqueta foreach var (lista) {... }

etiqueta foreach var (lista) {... } continue {... }

if (condición) {... }

if (condición) {... } else {... }

if (condición) {... } elsif (condición) {... } else {... }

Cuando se controla a una sola declaración, los modificadores de declaración proporcionan una sintaxis más ligera:

declaración if condición;

declaración unless condición;

declaración while condición;

declaración until condición;

declaración foreach lista;

En el caso del *if* en PERL existen 2 variantes en las expresiones que usan en las condiciones de comparación:

1.- Comparación númerica:

== igual que...

!= distinto de...

< menor que...

> mayor que...

>= mayor o igual que...

<= menor o igual que...

2.- Comparación alfanumérica, literal o de textos.

- eq igual que...

- ne distinto de...

- lt menor que...

- gt mayor que...

- ge mayor o igual que...

- le menor o igual que...

En el caso de que queramos repetir una acción o grupo de acciones dependiendo de una condición, podemos usar las estructuras repetitivas for...., foreach...y while...Todas estas instrucciones producen los denominados bucles.

Ej:

```
@list = ("var1", "var2", "var3", "var4");

foreach $var_temp (@list){

print "$var_temp\n";

}
```

Subrutinas.

Las subrutinas se definen con la palabra clave *sub* e invocadas simplemente nombrándolas. Si la subrutina en cuestión no ha sido todavía declarada, es necesario, para el proceso de parseo, poner los paréntesis.

```
rutina();   # paréntesis necesarios aquí...

sub rutina {... }

rutina;    #... pero no aquí
```

Los parámetros de una subrutina no necesitan ser declarados, ni en número ni en tipo; de hecho, pueden variar en cada llamada.

Cualesquiera de los argumentos pasados están disponibles para la subrutina en el array especial @_. Los elementos de @_ son asociados a los argumentos actuales.

Un modismo común es asignar @_ a una lista de variables con nombres:

Ej: my($x, $y, $z) = @_;

La palabra clave *my* indica que las siguientes variables están léxicamente embebidas en el bloque que las contienen.

Expresiones regulares.

El lenguaje Perl incluye una sintaxis especializada para escribir expresiones regulares y el intérprete contiene un motor para emparejar strings con expresiones regulares. El motor de expresiones regulares usa un algoritmo de Vuelta Atrás (backtracking), extendiendo sus capacidades desde el simple emparejamiento de patrones simples con la captura y sustitución de strings.

El operador m// (empareja) permite comprobar un emparejamiento por medio de una expresión regular. (Para abreviar, el precedente m puede ser omitido.) En el caso más simple, una expresión como:

```
$x =~ m/abc/  #Evalúa a verdadero si y sólo si $x empareja con la expresión regular abc.
```

El operador s/// (sustitución) especifica una operación de búsqueda y reemplazo:

```
$x =~ s/abc/aBc/; # Convierte la b en mayúscula.
```

Si buscamos una palabra determinada solo al principio de una cadena se usa el caracter:^.

```
$x =~ /^Hola/
```

Si buscamos una determinada palabra solo al final de una cadena se usa el caracter: $.

```
$x =~ /$Mundo/
```

Una combinación de solo ambos metacaracteres anteriores: ^$, significa línea en blanco.

```
$cadena =~ /^$/
```

En ocasiones es preferible usar una variable implícita $_ para realizar las búsquedas, de este modo no es necesario incluir la variable en la comparación.

```
$_="Hola Mundo";

if ( /Hola/){ };
```

Éstos son algunos de los metacaracteres que se usan, hay otras combinaciones de caracteres que actúan como patrones con distintos significados y que no nos vamos a parar en su explicación.

Funciones aplicadas a expresiones regulares.

- Split. Esta función se basa en la división o troceado de una cadena en distintos campos, usando como delimitador un carácter dado y que formará parte de una expresión regular de búsqueda. La función devuelve una matriz o array con los distintos campos obtenidos.

- Join. Esta función es contraria a la anterior y consigue componer un registro con distintos campos, usando como delimitador de cada campo un carácter dado. La función devuelve una cadena con campos delimitados.

2.3.2.4 Interfaces gráficos: GTK y Glade

2.3.2.4.1 Bibliotecas GTK+.

GTK es un grupo importante de bibliotecas o rutinas para desarrollar interfaces gráficas de usuario (GUI) para principalmente los entornos gráficos GNOME, XFCE y ROX de sistemas Linux.

GTK es la abreviatura de GIMP toolkit (conjunto de rutinas para GIMP). Es software libre (bajo la licencia LGPL), multiplataforma y parte importante del proyecto GNU.

Inicialmente fue creado para desarrollar el programa de manejo de imágenes GIMP, sin embargo actualmente es muy usado por muchos otros programas en los sistemas GNU/Linux. Cabe mencionar que Qt es una alternativa a GTK que también es muy utilizada (en el entorno KDE, por ejemplo).

GTK+ se ha diseñado para permitir programar con lenguajes como C, C++, Java (Sun), Perl o Python.

Actualmente su última versión es GTK+ 2, con una cantidad importante de mejoras respecto a la primera versión, aunque sin embargo, no es compatible con ella.

Bibliotecas de GTK+:

GTK+ se basa en tres bibliotecas:

- Glib es una biblioteca de bajo nivel estructura básica de GTK+ y GNOME. Proporciona manejo de estructura de datos para C, portabilidad, interfaces para funcionalidades de tiempo de ejecución (runtime) como ciclos, hilos, carga dinámica o un sistema de objetos.

- Pango es una biblioteca para el diseño y renderizado de texto, hace hincapié especialmente en la internacionalización. Es el núcleo para manejar las fuentes y el texto de GTK+2.

- ATK es una biblioteca para crear interfaces con características de una gran accesibilidad muy importante para personas discapacitadas o minusválidas. Pueden usarse útiles como lupas de aumento, lectores de pantalla, o entradas de datos alternativas al clásico teclado o ratón de ordenador.

Las librerías que normalmente son enlazadas son:

- La librería GTK (-lgtk), la librería de widgets que se encuentra encima de GDK.

- La librería GDK (-lgdk), el wrapper de Xlib.

- La librería glib (-lglib), que contiene diversas funciones. GTK está construida encima de glib por lo que siempre se usará.

- La librería Xlib (-lX11) que es usada por GDK.

- La librería Xext (-lXext) contiene código para pixmaps de memoria compartida y otras extensiones.

- La librería matemática (-lm). Es usada por GDK para diferentes cosas.

Un proyecto nuevo escrito en GTK se inicia con la siguiente línea:

```
gtk_init (&argc, &argv);
```

Llama a la función gtk_init (gint *argc, gchar *** argv) responsable de `arrancar' la librería y de establecer algunos parámetros (como son los colores y los visuales por defecto).

Teoría de señales y respuestas:

GTK es un toolkit (conjunto de herramientas) gestionadas mediante eventos.

El control se transfiere mediante “señales”. Cuando sucede un evento, como por ejemplo la pulsación de un botón, se emitirá la señal apropiada por el widget pulsado. Así es como GTK proporciona la mayor parte de su utilidad. Hay un conjunto de señales que todos los widgets heredan, como por ejemplo “destroy” y hay señales que son específicas de cada widget, como por ejemplo la señal “toggled” de un botón de selección.

Al crear una aplicación normalmente se quiere que haya más de un widget por ventana. Aquí es donde entra el empaquetamiento. Normalmente para empaquetar se usan cajas. Éstas son widgets invisibles que pueden contener nuestros widgets de dos formas diferentes, horizontal o verticalmente. Para crear una caja horizontal llamamos a gtk_hbox_new() y para las verticales gtk_vbox_new().

Usando estas funciones podemos ir metiendo widgets con una justificación a la izquierda o a la derecha y además podemos mezclarlas de cualquier manera para conseguir el efecto deseado.

Vamos a explicar algunos de los widgets usados en nuestra aplicación:

Libros de notas (Notebooks):

El widget Notebook es una colección de “páginas” que se solapan las unas a las otras, cada una con un contenido diferente. Este widget se ha vuelto cada vez más común últimamente en la programación de interfaces gráficos de usuario (GUI en inglés), y es una buena forma de mostrar bloques de información similar que necesitan aparecer de forma separada.

El widget árbol:

El propósito del widget GtkTree es mostrar datos organizados de forma jerárquica. Este widget es un contenedor vertical para los widgets del tipo GtkTreeItem. GtkTree en sí mismo no es muy diferente de GtkList - ambos están derivados directamente de GtkContainer, y los métodos GtkContainer funcionan igual en los widgets GtkTree que en los GtkList. La diferencia es que los widgets GtkTree pueden anidarse dentro de otros widgets GtkTree.

El widget GtkTree tiene su propia ventana, y tiene por defecto un fondo de color blanco, como GtkList. La mayoría de los métodos de GtkTree funcionan igual que sus correspondientes de GtkList. Sin embargo, GtkTree no está derivado de GtkList, por lo que no puede intercambiarlos.

Monitorizando la E/S

Otra característica de GTK, es la habilidad que tiene de comprobar datos en un descriptor de fichero (tal y como se devuelven por open(2) o socket(2)). Esto es especialmente útil para las aplicaciones de red.

2.3.2.4.2 Glade

Glade es una herramienta de desarrollo visual de aplicaciones mediante GTK/GNOME. Tiene licencia GPL.

Aunque tradicionalmente se ha utilizado de forma independiente, está integrado en el recientemente liberado Anjuta 2, una plataforma integrada de desarrollo (IDE).

Glade puede crear la interfaz de usuario de las aplicaciones de dos formas diferentes, o bien generando código fuente, o bien cargando dinámicamente un fichero XML de descripción de la interfaz en tiempo de ejecución. Cualquiera de las dos alternativas está disponible para una gran cantidad de lenguajes de programación.

Widgets

Los contenedores son widgets de utilidad para el programador. El usuario nunca se enterará de su existencia. Sin embargo, conocer bien su funcionamiento incidirá en una mayor productividad del programador, puesto que son estos widgets los encargados de distribuir los widgets visuales en una ventana.

Libglade y las bibliotecas de GNOME

Para integrar los widgets que se construyen sobre GTK+, sólo es necesario realizar la inicialización a través de “gnome_program_init” en vez de “gtk_init”.

La idea es ir colocando los diferentes componentes (o widgets) tal y como queramos nuestra Interfaz con el usuario, indicando la posición y las diferentes características de cada elemento. Una vez creado el diseño de la aplicación, Glade nos creará la estructura básica del código, restando la programación de las llamadas de cada widget y del código del programa en sí.

Glade permite al desarrollador el poder crear rápida y eficientemente una aplicación visual para después poder concentrarse en la implementación real del programa, en lugar de estar bloqueado por cuestiones de la interfaz del usuario.

Iniciando un aplicación con Glade

Al iniciar el programa Glade se puede observar que aparecen tres ventanas:

1. Ventana principal,

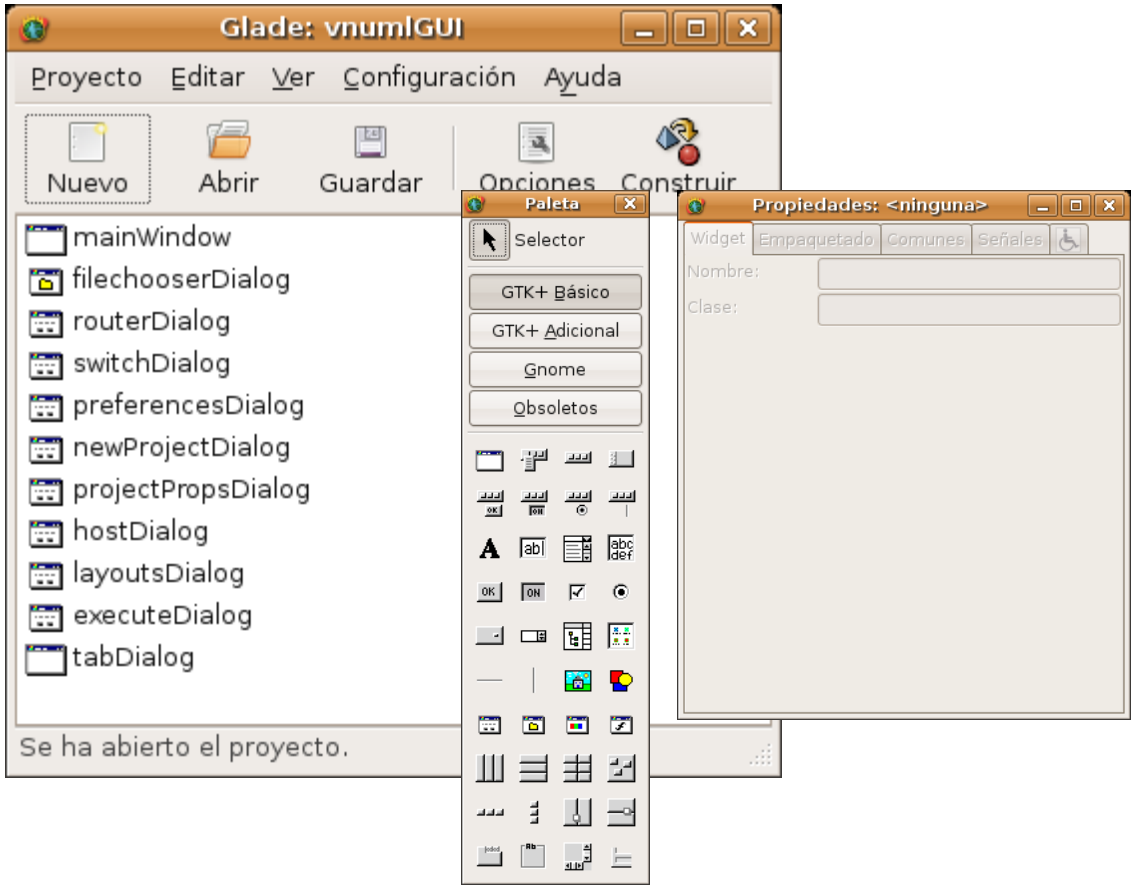
2. Paleta de Widgets y

3. Editor de propiedades.

La "Ventana principal" muestra los widgets que vayamos añadiendo a nuestra aplicación. Por widget entenderemos cualquier componente que queramos incluir en la interfaz, por ejemplo: Botones, Barra de menú, Barras de progreso, Cuadro de diálogo, etc.

La "Paleta de Widgets" permite seleccionar qué elementos agregaremos a nuestra aplicación.

El "Editor de propiedades" se activa cuando un widget esta seleccionado y permite modificar sus atributos y la definición de señales.



Eventos

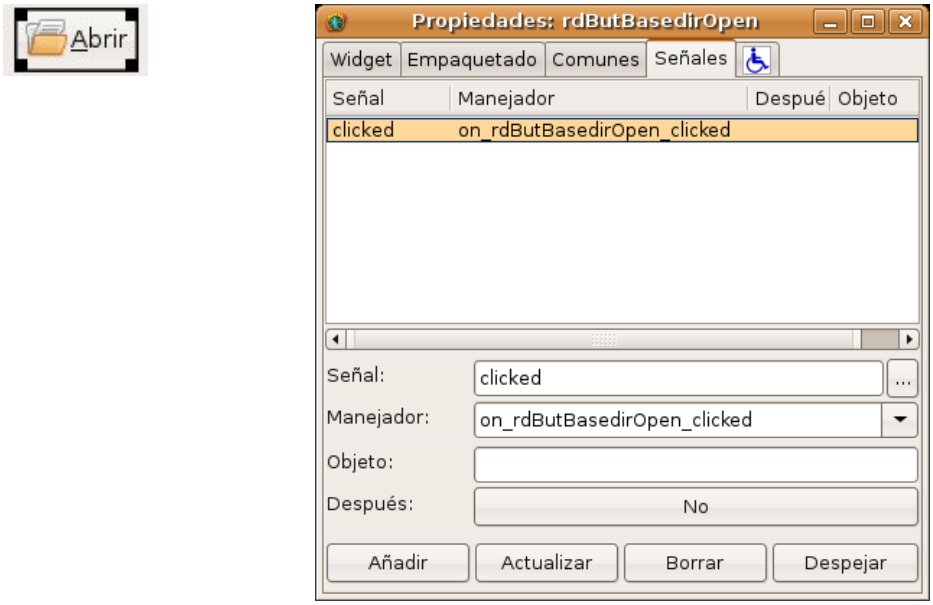
Cada uno de estos widgets está asociado al menos a un evento, así que vamos a dedicar unas líneas a este concepto.

Siempre que pulsamos una tecla o presionamos algún botón del ratón se produce un evento y se emite una señal. Los widgets de la interfaz deben responder a uno o más eventos.

Por ejemplo, podemos diseñar un elemento de un cuadro de diálogo que se llame "Abrir", y precisamente queremos que la aplicación responda de una determinada manera al pulsar sobre dicho elemento. Hay muchísimos eventos y señales, y la mayoría de ellos van a ser ignorados por la aplicación (por ejemplo, el teclado no va a jugar ningún papel en la aplicación que vamos a desarrollar aquí, así que las pulsaciones de las teclas serán simplemente ignoradas). Si queremos que un elemento de la interfaz responda a un determinado evento, debemos escribir una función que indique al programa la respuesta que queremos que ofrezca a tal acción. A estas funciones se les llama retrollamadas (callbacks).

Para ello, primero debemos crear un manejador de señal (signal handler) para un evento del cual queramos que suceda algo. Una vez que es creado el manejador de señal hacemos que corresponda a una función de respuesta a señal (callback). Esta función es lo que realmente se ejecuta una vez que ha sido hecha la llamada.

Para el ejemplo del botón "Abrir" de uno de los Cuadros de Diálogo de nuestra aplicación, la siguiente imagen sería lo que aparecería en la ventana Editor de Propiedades de Glade:



A este botón se le ha asociado la señal clicked, que será el evento al cual queremos que responda la acción de pulsar el botón. El manejador de señal, en este ejemplo, se llama "on_rdButBasedirOpen_clicked".

En el código fuente de nuestra aplicación, encontraremos la siguiente función callback:

```
sub on_rdButBasedirOpen_clicked() {
 my ($widget, $event) = @_;
 &debug("$widget received $event");

 $STORE{'GLADEXML'}->get_widget('filechooserDialog')->set_select_multiple(FALSE);
 $STORE{'GLADEXML'}->get_widget('filechooserDialog')->set_action('select-folder');

 $STORE{'GLADEXML'}->get_widget('filechooserDialog')->
set_transient_for($STORE{'GLADEXML'}->get_widget('mainWindow'));
 my $retval = $STORE{'GLADEXML'}->get_widget('filechooserDialog')->run();
 $STORE{'GLADEXML'}->get_widget('filechooserDialog')->hide();

 if ($retval eq 'ok') {
 my $dir = $STORE{'GLADEXML'}->get_widget('filechooserDialog')->get_filename();
 $STORE{'GLADEXML'}->get_widget('rdEntBasedir')->set_text($dir);
 }
 return TRUE;
}
```

Este es el código que corresponde a lo que queremos que el programa haga cuando presionemos dicho botón.

3. MEJORAS EN LA HERRAMIENTA VNUMLGUI

3.1 Cambios relacionados con la sintaxis de la DTD.

Como ya hemos mencionado anteriormente, la herramienta VNUML (Virtual Network User Mode Linux), es un sistema de propósito general de código abierto para la simulación de entornos de red basados en GNU/Linux. Entre sus múltiples usos se encuentran la prueba de aplicaciones en entornos controlados o el prototipado rápido de escenarios de red.

El escenario que se desea simular mediante VNUML se describe usando XML.

VNUMLGUI traduce el archivo XML de forma que lo entiende y lo interpreta dibujando sus elementos y haciendo que todos los componentes del interfaz tomen los valores adecuados según se especifican en el fichero XML.

Ya que el lenguaje de las simulaciones es XML, VNUML tiene definida la descripción de estructura y sintaxis de los documentos XML, la DTD.

VNUMLGUI es una aplicación para VNUML en sus versiones 1.5 y 1.6. Debido a las posteriores actualizaciones de VNUML, VNUMLGUI se ha quedado obsoleto y poco ventajoso para los usuarios de VNUML.

La actualización de VNUML a la versión 1.7.0-1 se basa especialmente en los cambios producidos en su DTD. Por ello la actualización del interfaz gráfico VNUMLGUI se reduce a estos cambios de la DTD, introduciendo nuevos elementos y/o modificando otros antiguos, para que cuando se procese un documento XML a través de VNUMLGUI el resultado del análisis de sus datos (mediante el parser) sea satisfactorio.

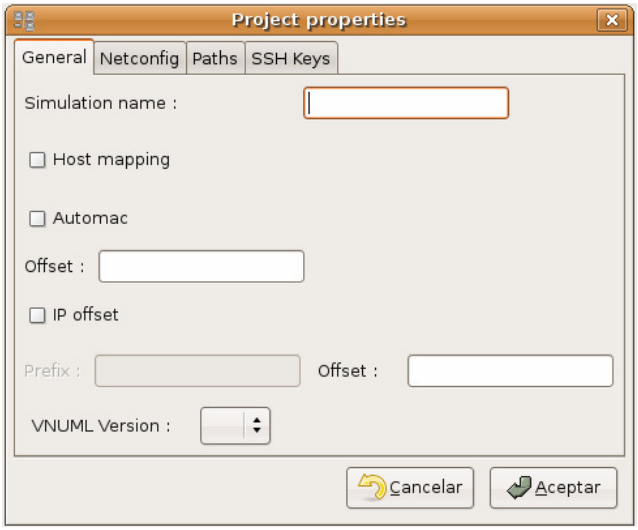
Las principales etiquetas de un archivo xml de VNUML son <global>, describe los elementos globales de la simulación, <net>, describe redes virtuales, <vm>, especifica las propiedades de las máquinas virtuales y <host>, describe la configuración del host.

No nos vamos a detener en la definición del significado de cada uno de los elementos de la DTD, ya que nuestro propósito no es el estudio de VNUML, lo único que nos importa en este capítulo es el contenido de la DTD: la declaración de los nombres de las etiquetas, los atributos, las restricciones...

A continuación se explica el proceso de actualización de VNULMGUI para la versión VNUML 1.7.0-1. Se describen las etiquetas que han sufrido algún cambio de actualización y que por consiguiente, producen algún cambio en el código fuente de VNUMLGUI.

3.1.1 Propiedades globales de los proyectos

La vista del interfaz original correspondiente a esta sección se correspondía con la siguiente ventana:



A continuación vamos a explicar de qué forma se ha ido modificando esta ventana.

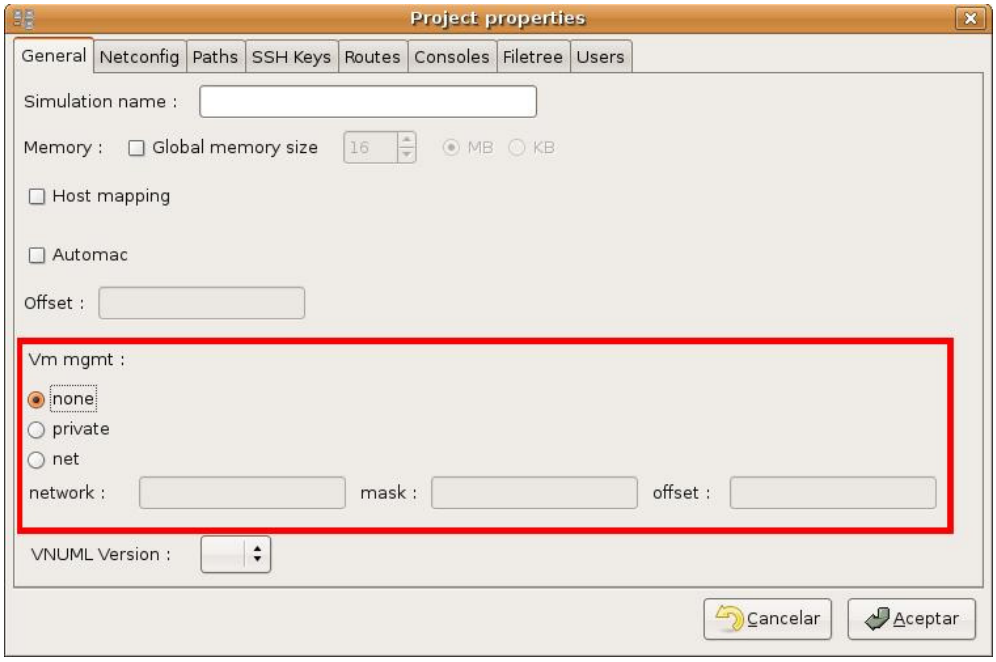
<global>:

- Se han eliminado todas las instancias de la etiqueta <ip_offset> en el código fuente y en el interfaz, debido a que en las posteriores versiones esta etiqueta desaparece definitivamente. En el interfaz se elimina la casilla “IP offset”, y los campos de texto “Prefix” y “Offset”.

- Surge una nueva etiqueta <ssh_version>, que proviene de lo que era el atributo “version” de la etiqueta <ssh_key>. Esta modificación no produce ningún cambio en el interfaz gráfico, pero sí en el código fuente de la aplicación. Al abrir un fichero existente xml, la rutina dedicada a ello, sabrá interpretar la existencia de esta nueva etiqueta.

- <vm_mgmt>. Los elementos pertenecientes a esta etiqueta existían en la ventana del interfaz Preferencias del menú Edit, pero debido a las necesidades surgidas a lo largo del desarrollo del proyecto, fue necesario incluir los componentes relativos a esta etiqueta en la sección Propiedades del menú Archivo.

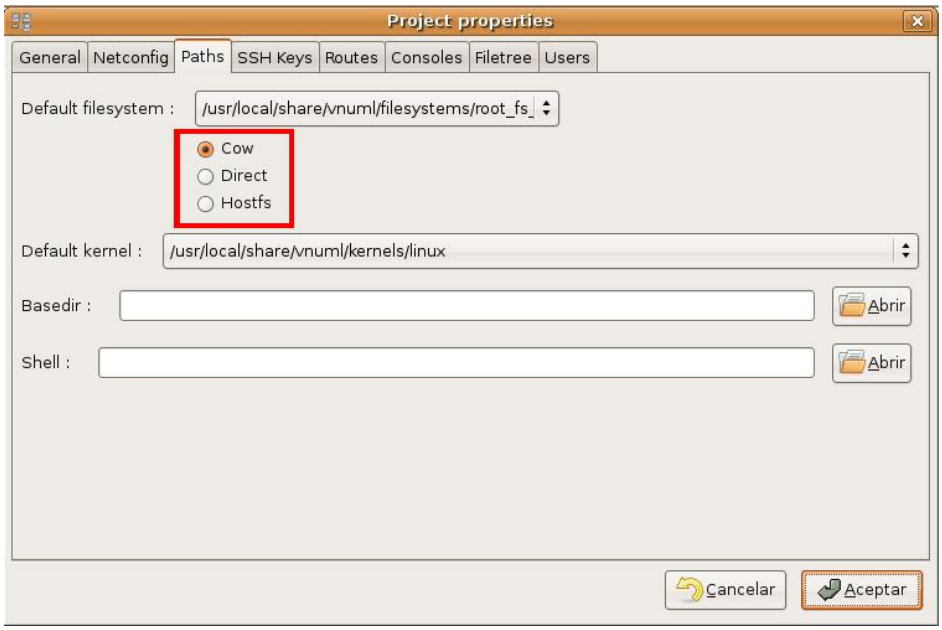
  Se ha realizado un arreglo en esta etiqueta de la ventana Preferencias del menú Edit. Cuando existía el valor vnuml.vm_mgmt_type en el archivo de configuración ~/.vnuml/config.xml, el campo de esta etiqueta se debía actualizar en el interfaz al abrir VNUMLGUI. Ésto no ocurría y como consecuencia siempre aparecía en el xml la etiqueta <vm_mgmt type = “none”>, a pesar de que type tuviese otro valor. Éste error ha sido solucionado.



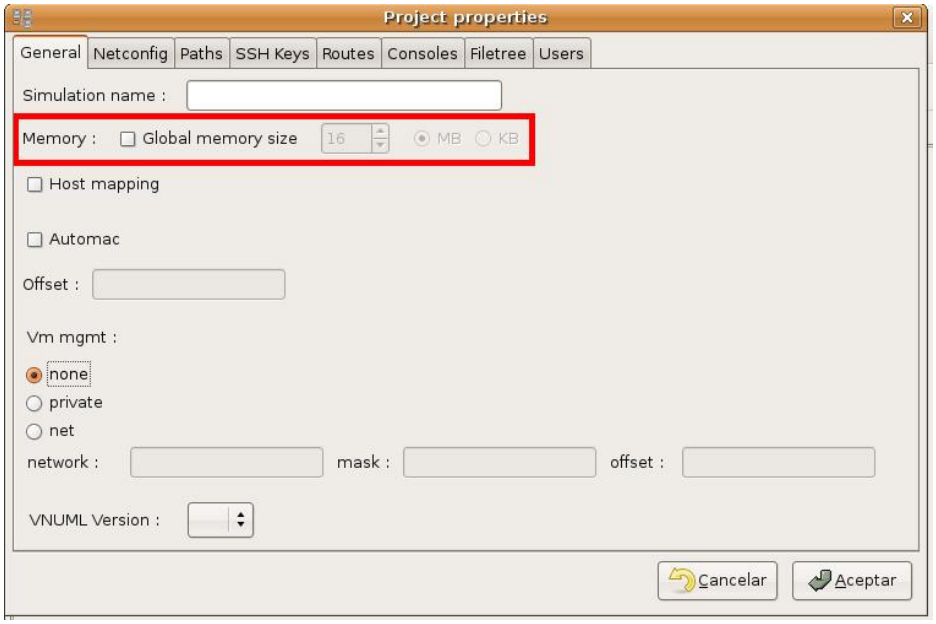
- <vm_defaults>. Esta etiqueta nueva describe los valores por defecto usados en todas las máquinas virtuales (especificadas individualmente por la etiqueta <vm>). Su estructura es bastante similar a la de <vm> y por tanto, a la ventana Virtual Machine Properties del interfaz. La existencia de esta nueva etiqueta ha hecho que se modifique considerablemente la apariencia de la ventana Propiedades del menú Archivo.

  Esta etiqueta contiene a su vez otras nuevas - en particular, <filesystem>, <mem>, <kernel>, <shell>, <basedir>, <mng_if>, <console>, <xterm>, <route>, <forwarding>, <user> y <filetree> - y cuyos componentes son incluidos en esta ventana:

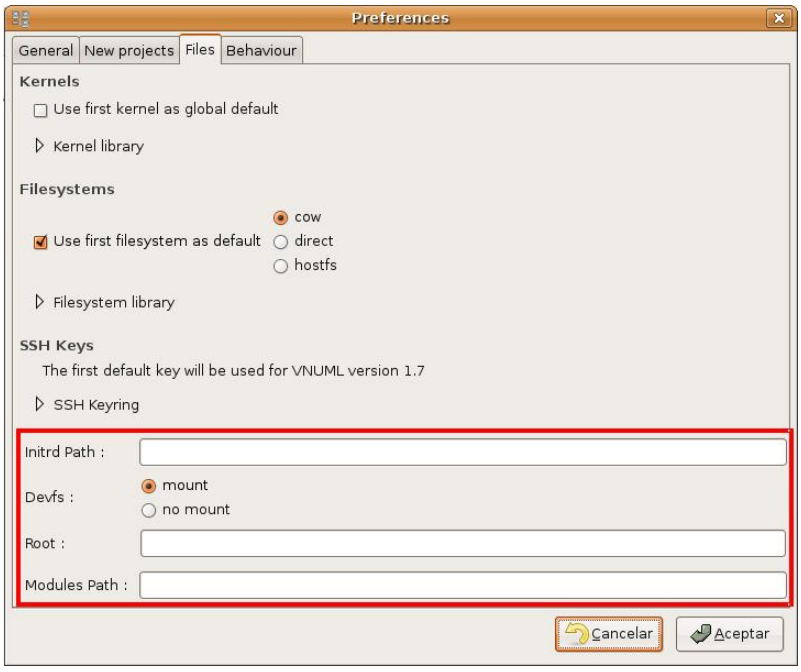
  - <filesystem>: Esta etiqueta posee tres atributos “direct”, “cow” y “hostfs”, en el interfaz están representados mediante componentes RadioButtons. En versiones anteriores existía el atributo “copy”, que ha sido eliminado.



- <mem>: Esta etiqueta existía en la ventana Preferencias del menú Edit, pero debido a las necesidades surgidas a lo largo del desarrollo del proyecto, fue necesario incluir los componentes relativos a esta etiqueta también en la ventana Propiedades del menú Archivo.



- <kernel>: Se han introducido tres nuevos atributos propios de esta etiqueta, “devfs”, “root”y “modules”. Estos nuevos elementos se han añadido en la ventana Preferencias del menú Edit, debajo del cuarto atributo de esta etiqueta que ya existía, “initrd”. En este atributo se ha hecho un pequeño arreglo, cuando existía el valor vnuml.kernel_initrd en el archivo de configuración ~/.vnumlgui/config.xml, el valor de éste debía aparecer en su campo correspondiente de la ventana Edit->Preferencias, ésto no ocurría y por tanto se ha solucionado.

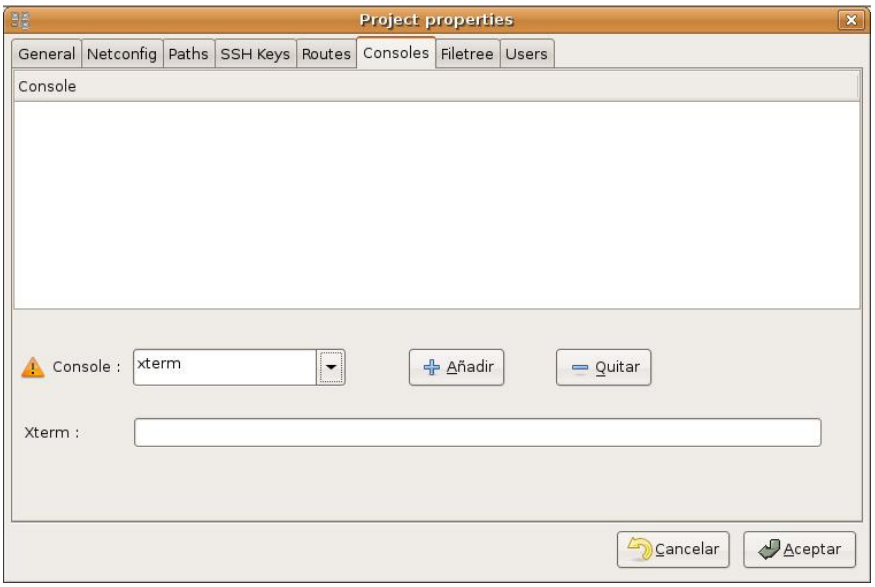


- <shell> y <basedir>: Estos dos componentes existían ya en la ventana Propiedades del menú Archivo, debido a que en la jerarquía del árbol XML existían bajo la etiqueta <globals> y no bajo <vm_defaults>, como consecuencia, el interfaz gráfico no se ha visto modificado, pero sí se han introducido cambios en el código fuente de VNUMLGUI, para que estos elementos sean procesados bajo la etiqueta <vm_default>.

- <mng_if>: Se han introducido cambios en el código fuente de VNUMLGUI, para que este elemento sea procesado bajo la etiqueta <vm_default>.

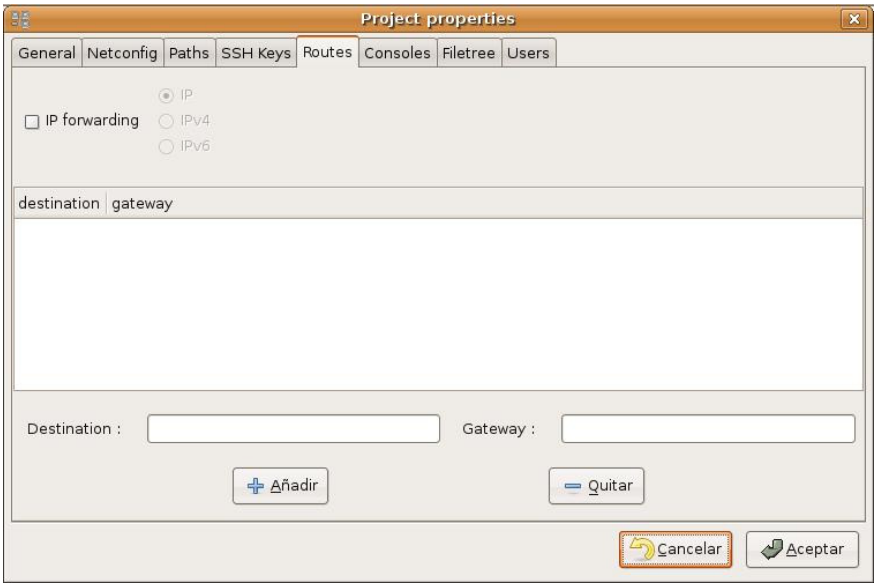
- <console>: En las versiones anteriores, este elemento era único, ahora es múltiple, por tanto esta etiqueta puede aparecer más de vez en un documento XML, ello ha supuesto crear una nueva pestaña en la ventana Propiedades, que incluya un componente FileTree para mostrar e ir añadiendo el contenido de esta etiqueta en sus sucesivas apariciones.

- <xterm>: Este elemento se añade al interfaz como una entrada de texto en la pestaña Consoles de la ventana Propiedades. Se habilita para poder escribir, unicamente cuando se ha seleccionado de la lista desplegable de console el elemento correspondiente a xterm.



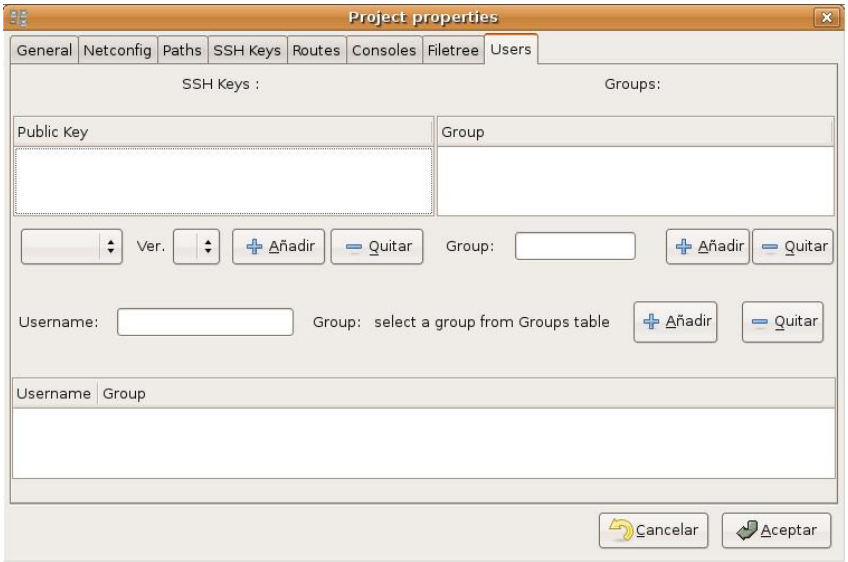
- <route>: Respecto al interfaz, se ha creado una nueva pestaña Routes en la ventana propiedades del menú Archivo, el contenido de esta pestaña es exactamente igual a la pestaña del mismo nombre que ya existía en la ventana Virtual Machine Properties. Respecto al código fuente, ha habido pequeñas modificaciones ya que los atributos que antes se llamaban “inet” e “inet6” ahora se llaman “ipv4” e “ipv6” respectivamente. Se han modificado estos datos para que el 'parser' no falle en el análisis del documento.

- <forwarding>: Este componente forma parte del interfaz Routes, por tanto ha sido incluído en la nueva pestaña Routes de la ventana Propiedades del menú Archivo.

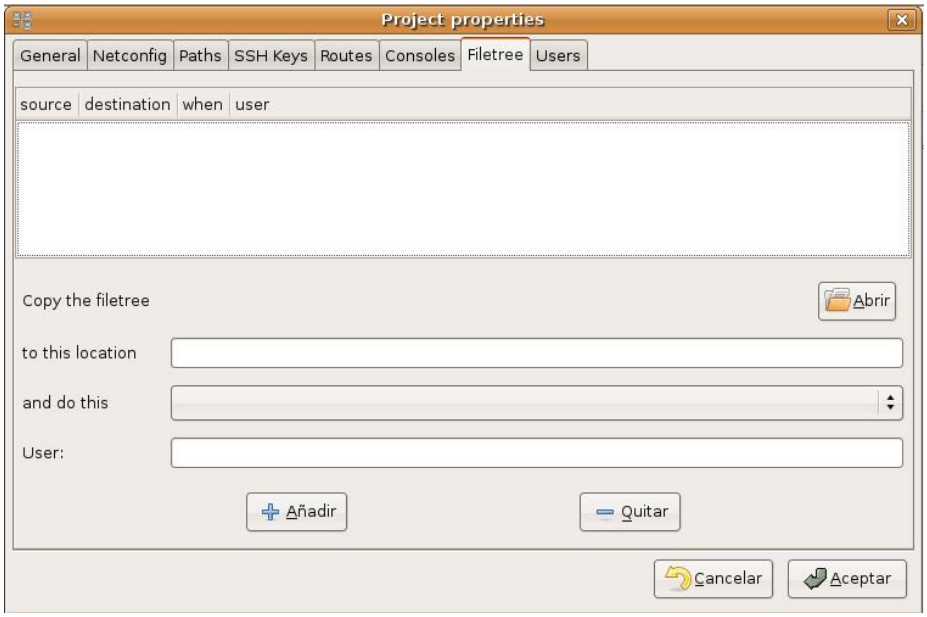


- <ipv6>: Esta etiqueta es subetiqueta de <forwarding>. La actualización de la DTD le ha afectado de modo que se ha introducido un nuevo atributo "mask". Estos cambios se reflejan en el código fuente, de manera que cuando se identifica una dirección ipv6 seguida de una barra (/) y luego la máscara, sepa interpretar esta información de máscara y guardarla en el atributo "mask".

- <user>: Este elemento es completamente nuevo en VNUMLGUI. Se ha creado una nueva pestaña Users en la ventana propiedades del menú Archivo que contiene todos sus componentes. Se ha modificado el código fuente considerablemente para el trato de esta etiqueta.

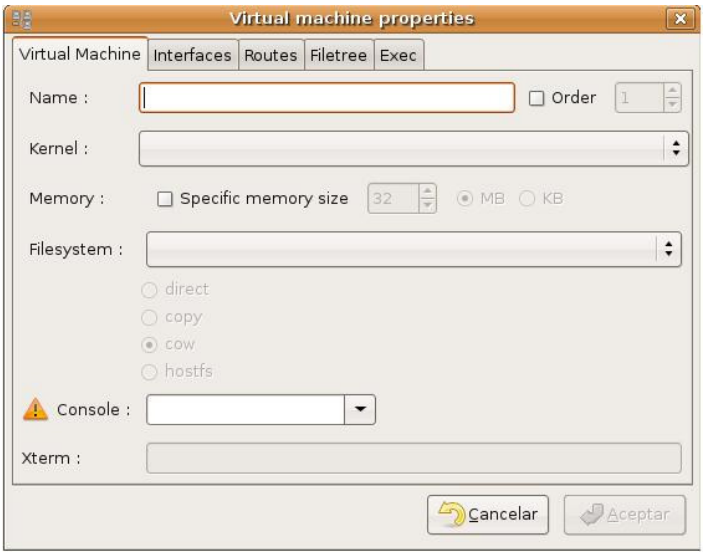


- <filetree>: Al igual que se ha hecho con Consoles y Routes, se ha creado una nueva pestaña Filetree en la ventana propiedades del menú Archivo, ya que en esta versión esta etiqueta es subetiqueta de <global>. El contenido de esta pestaña es exactamente igual a la pestaña del mismo nombre que ya existía en la ventana Virtual Machine Properties, con una excepción, y es que se le ha añadido además un campo de texto para escribir el contenido del nuevo atributo "user" de esta etiqueta.



3.1.2 Propiedades de las Máquinas Virtuales

La vista del interfaz original correspondiente a esta sección se correspondía con la siguiente ventana:



Seguidamente explicamos cómo se ha ido modificando esta ventana.

<vm>:

- <filesystem>: Esta etiqueta es la misma que la mencionada anteriormente, <filesystem>, subetiqueta de <global>, pero en este caso es subetiqueta de <vm>, por tanto también posee tres atributos “direct”, “cow” y “hostfs”, representados mediante componentes RadioButtons, y cuyo atributo “copy” ha sido eliminado.

- <shell> y <basedir>: Estos dos componentes han sido actualizados de forma que en la jerarquía del árbol XML aparecen bajo la etiqueta <vm>, y bajo <vm_defaults> como hemos mencionado anteriormente. Como consecuencia de ésto, en el interfaz gráfico se han incluido dos nuevas entradas de texto en los cuales se escribe el contenido de estos elementos.

- <console> y <xterm>: La etiqueta obsoleta <boot>, que tenía como subetiquetas <con0> y <xterm>, ha sido sustituída por las etiquetas <console> y <xterm>, que trabajan de la misma forma que las etiquetas del mismo nombre que aparecen en la jerarquía del árbol XML bajo <vm_defaults> y que se han explicado anteriormente. De la misma forma se ha creado una nueva pestaña Consoles que incluye un componente FileTree para mostrar e ir añadiendo el contenido de esta etiqueta <console> en sus sucesivas apariciones, ya que puede aparecer múltiples veces en el documento XML. También se añade en esta pestaña una entrada de texto para el contenido de <xterm>. Esta entrada se habilita para poder escribir, únicamente cuando se ha seleccionado de la lista desplegable de console el elemento correspondiente a xterm.

  Debido a la inclusión de esta nueva pestaña, se han eliminado en la primera pestaña de la ventana Virtual Machina Properties, los componentes antiguos del interfaz que hacían referencia a la etiqueta <boot>, como una lista desplegable para elegir el valor de <con0> y un cuadro de texto para el valor de <xterm>.

  No mostramos la imagen para la pestaña Consoles ya que es idéntica a la del cuadro de diálogo Project Properties, explicada anteriormente.

- <forwarding>: Esta etiqueta ha sufrido una modificación. Cuando se creaba una maquina virtual nueva, y no se marcaba la casilla “IP forward”, una vez que se pulsaba el boton “Aceptar”, esta casilla se marcaba automaticamente y se escribía en el xml: <forwarding type="ip"/>. Esta situación de que apareciese éste valor por defecto se ha eliminado ya que a veces resultaba molesto.

- <filetree>: A esta etiqueta se le ha añadido un nuevo atributo, “user”, y como consecuencia de ésto, se ha incluido un campo de texto, en la pestaña “Filetree” para introducir el contenido de este atributo.

  No mostramos ninguna imagen de la pestaña “Filetree” ya que es idéntica a la de la ventana Propiedades del menu Archivo.

- <exec>: Esta es la misma situación que el caso anterior. Se ha incluido en la pestaña “Exec” un nuevo campo de texto “User”, para introducir el contenido del nuevo atributo “user” que se le ha añadido a esta etiqueta.

  Esta modificación se ha realizado en la ventana Virtual Machine Properties y en la ventana Host Properties.

- <user>: Al igual que en la ventana Propiedades del menu Archivo, se ha creado una nueva pestaña Users de la misma apariencia. También se ha modificado el código fuente en gran medida para el trato de esta etiqueta.

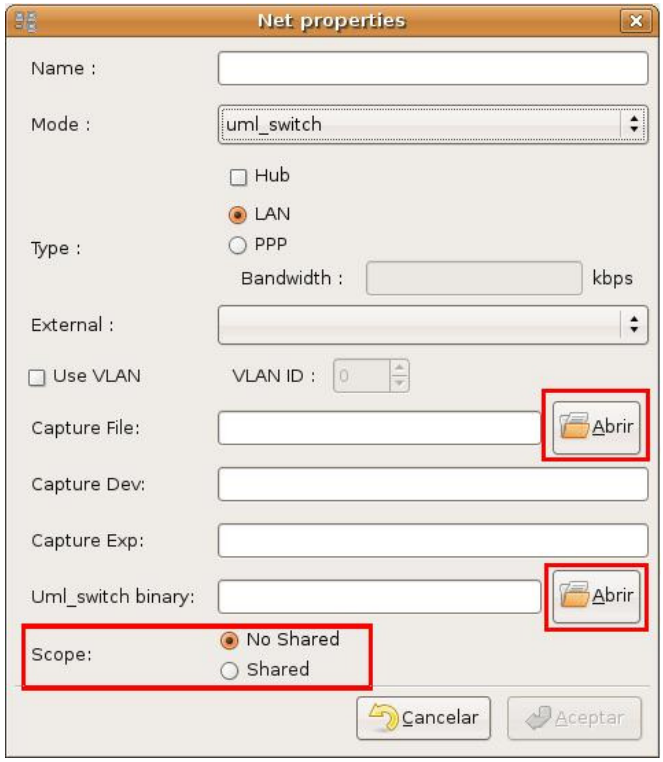
  No mostramos ninguna imagen ya que es idéntica a la de la ventana Propiedades del menu Archivo.

3.1.3 Propiedades de las Redes

- <net>:
    - "scope": Se ha incluido este nuevo atributo con dos valores posibles: "shared" y "no-shared". En la ventana Net Properties se introducen estos elementos en forma de componentes RadioButtons, tal y como muestra la imagen más abajo.
    - "mode": Este atributo ha cambiado sus propiedades a Required, de modo que el boton Aceptar de la ventana Net Properties que antes se habilitaba cuando se introducía un nombre para la red (atributo "name"), ahora se habilitará cuando, aparte de introducir un nombre, se elija también un modo (atributo "mode").

En este cuadro de diálogo, Net Properties, se han incluido también dos botones de apertura de ficheros con el objetivo de facilitar la escritura de rutas en los campos texto "Capture File" y "Uml_switch binary".

Respecto a los campos "Capture File", "Capture Dev", "Capture Exp" y "Uml_switch binary", se ha tenido que incluir la situación de que se borre el contenido de estos campos si el modo, "mode", que se selecciona no es "uml_switch", ya que cuando se introducía algo en estos campos habiendo elegido el modo "uml_switch", y posteriormente se cambiaba el modo a "virtual_bridge", se deshabilitaban estos campos sin limpiarlos y como consecuencia, al construir el escenario saltaba un error ya que existían datos en esos campos que no eran propios del modo virtual_bridge.



3.1.4 Implementación

Como ya hemos mencionado antes, VNUMLGUI genera el documento XML para la especificación de las simulaciones de VNUML. Debido al cambio y modificaciones producidas en la DTD de VNUML, VNUMLGUI tiene que generar y escribir correctamente el documento XML siguiendo esta DTD.

Para ello se ha tenido que modificar la rutina “dump_xml”, encargada del volcado del documento XML. Este método está particularizado para los tipos router, uml_switch y host.

Otra de las funciones de VNUMLGUI es el procesado de un fichero XML existente, por lo que un usuario de VNUMLGUI puede abrir un documento XML, en el cual se define una determinada simulación, leerlo, modificarlo mediante su interfaz gráfica, y volver a escribirlo correctamente. Para ello se han tenido que introducir modificaciones en la rutina “open_file”.

Una variable global de notable importancia y que por ello debemos mencionar es el hash STORE. Es una variable global con estrutura de hash que almacena los objetos en tiempo de ejecución y la configuración de la especificación.

Para facilitar una posterior actualización de VNUMLGUI, hemos tomado la decisión de establecer una regla o estipulación de nombres que afectan a los atributos y etiquetas definidos en la DTD, y que guardamos en la variable STORE:

Regla para las etiquetas:

- $STORE{'SIMULATION'}{'TAG'}

Regla para los atributos:

- $STORE{'SIMULATION'}{'TAG_ATTRIB'}

Por ejemplo:

La etiqueta <automac> y su atributo “offset” están definidos en la DTD de la siguiente manera:

<!ELEMENT automac EMPTY>

<!ATTLIST automac offset CDATA “0”>

Estos elementos se guardan en la estructura STORE, siguiendo las reglas establecidas, de la siguiente manera:

$STORE{'SIMULATION'}{'AUTOMAC'}

$STORE{'SIMULATION'}{'AUTOMAC_OFFSET'}

Al establecer esta convención de nombres, además de añadir nuevos nombres, se tuvo que modificar muchos de los nombres de etiquetas y atributos que ya existían en el código de VNUMLGUI, y adaptarlos siguiendo esta regla.

En vista a futuras actualizaciones de la herramienta VNUML, los anteriores desarrolladores de VNUMLGUI, utilizaban el módulo DataHandler del paquete VNUML en el método 'open_file', para que a la hora de procesar un documento XML y analizarlo el resultado fuera lo más satisfactorio posible y que estas actualizaciones de VNUML afectaran mínimamente al funcionamiento de VNUMLGUI. De este modo, a la hora de implementar, se ha seguido la misma política, pero haciendo pequeñas modificaciones.

Estas modificaciones están relacionadas con los atributos opcionales y por defecto de la DTD. Es decir, a la hora de abrir un documento XML existente, se añadían etiquetas y/o atributos sin que el usuario hubiese indicado su aparición explícita. Esto era debido a que el módulo DataHandler leía estos elementos y si no existían en el XML, les asignaba un valor por defecto.

Por ejemplo:

```
my $dh = new DataHandler(...);
my $automac = $dh->get_automac_offset();
if (length($automac) > 0) {
 $STORE{'SIMULATION'}{'AUTOMAC'} = TRUE;
 $STORE{'SIMULATION'}{'AUTOMAC_OFFSET'} = $automac;
}
else {
  $STORE{'SIMULATION'}{'AUTOMAC_OFFSET'} = 0;
}
```

Este fragmento de código hacía que siempre apareciese en el XML por defecto la etiqueta <automac offset=0> si no se había definido nungún otro valor.

Esto podía resultar molesto, ya que si el usuario no había indicado que apareciera esta etiqueta, al abrir el XML con VNUMLGUI y volverlo a guardar, se incluían estas pequeñas modificaciones sin consentimiento del usuario. Por lo tanto se modificó el código de esta forma:

```
my $automac = $doc->getElementsByTagName("automac");
if ($automac->getLength > 0) {
 $STORE{'SIMULATION'}{'AUTOMAC'} = TRUE;
 my $offset = $dh->get_automac_offset();
 if (length($offset) > 0) {
  $STORE{'SIMULATION'}{'AUTOMAC_OFFSET'} = $offset;
 }
 }
```

En este fragmento, si una etiqueta <automac> no existe en el XML, no se incluye, y si existe se lee su atributo offset mediante la función "get_automac_offset()" del módulo DataHandler, y toma su valor correspondiente.

Modificaciones como ésta se han realizado para otras etiquetas como por ejemplo <netconfig>, <tun_device>, <shell>, <mem>, <forwarding>.

Proceso de implementación.

En esta sección se detallará el proceso de implementación de alguna de las actualizaciones explicadas anteriormente. Se mostrará tanto las modificaciones realizadas en el código perl de VNUMLGUI como las realizadas en el interfaz gráfico.

Son muchos los ejemplos que podríamos mostrar, como podrían ser la creación del interfaz de la etiqueta <routes>, <filetree>, etc. u otro mas completo como el de la etiqueta <users>, pero ya que éste tiene bastantes widgets repetidos como botones, cuadros de texto, árboles, etc. nos vamos a decantar por describir el proceso de creación de un interfaz más sencillo como podría ser el de las etiquetas <console> y <xterm>. Como el interfaz referente a estas etiquetas aparece en dos ventanas de VNUMLGUI, la de Propiedades de Projecto del menu Archivo, y en la ventana Propiedades de las Máquinas Virtuales, se va a definir el proceso de implementación de la primera ventana.

El primer paso que realizamos es crear la apariencia del interfaz mediante la herramienta Glade.

Todos los widgets relativos a las etiquetas <console> y <xterm> se incluyen en el cuadro de diálogo Project Properties, cuyo widget se llama “projectPropsDialog”.

Se opta por incluir una nueva pestaña (widget notebook) en este cuadro de diálogo, con una etiqueta de nombre “Consoles”, para distinguirla del resto de las pestañas de la ventana.

Para introducir todos los demás elementos del interfaz dentro de este widget, creamos otro de la clase GtkVBox para posicionar el resto de widgets.

Esta etiqueta puede ser múltiple, es decir, se pueden especificar varias consolas con un número 'id' diferente:

Ejemplo:
```
<console id="0">xterm</console>
<console id="1">tty:/dev/tty8</console>
<console id="2">pts</console>
```

Debido a esta propiedad de multiplicidad, se va a incluir un árbol, widget de la clase GtkTreeView, que muestre todas las consolas especificadas hasta el momento. Llamamos a este widget “ppdTrwConsole”. Este widget se posiciona sobre otro de la clase GtkScrolledWindow, que proporciona las barras de desplazamiento de la sección de GtkTreeView, para poder visualizar todos los elementos que se van a incluyendo en el árbol.

En otro widget de la clase GtkHBox, se incluyen otros widgets más como son:

1. Un widget de la clase GtkEventBox, llamado “ppdEbConsoleWarn”, cuya función es mostrar una sugerencia cuando se posiciona el ratón sobre él.

2. Sobre este widget, se crea otro de la clase GtkImage, de nombre “ppdGiConsoleWarn”, éste muesta un pequeño icono de warning.

3. Al lado se incluye un widget de clase GtkLabel, cuya etiqueta es “Console:”.

4. Se crea un widget de la clase GtkComboBoxEntry, llamado “ppdCbeConsole”, encargado de mostrar la lista de consolas que podemos seleccionar para incluir en la especificación.

5. Al lado de este widget se incluye otro de la clase GtkHButtonBox, en el que vamos a introducir los dos botones siguientes, Añadir y Eliminar.

6. Sobre este widget creamos los widgets “ppdButConsoleAdd” y “ppdButConsoleRem” de la clase GtkButton, encargados de añadir o eliminar entradas en el árbol “ppdTrwConsole”.

Se incluye otro widget más de la clase GtkHBox, donde se insertan:

- Una etiqueta de clase GtkLabel, que muestra la palabra “Xterm”.
- Un widget de clase GtkEntry, llamado “ppdEntRtXterm”, que representa un campo de texto donde se introduce el contenido de la etiqueta <xterm>.

Los widgets relativos a los botones de añadir o eliminar, “ppdButConsoleAdd” y “ppdButConsoleRem”, anteriormente nombrados, tienen asociado un manejador de señal “on_ppdButConsoleAdd_clicked” y “on_ppdButConsoleRem_clicked” respectivamente. En el fichero perl del código fuente de VNUMLGUI se introduce los siguientes fragmentos de código:

```
sub on_ppdButConsoleAdd_clicked() {
 my ($widget, $event) = @_;
 &debug("$widget received $event");

 my $console = $STORE{'GLADEXML'}->get_widget('ppdCbeConsole')->
get_model->get($STORE{'GLADEXML'}->get_widget('ppdCbeConsole')->get_active_iter);

 push(@{$STORE{'GLADEXML'}->get_widget('ppdTrwConsole')->{'data'}},
 [ $console ]);

 return TRUE;
}

sub on_ppdButConsoleRem_clicked() {
 my ($widget, $event) = @_;
 &debug("$widget received $event");

 my @selected = $STORE{'GLADEXML'}->get_widget('ppdTrwConsole')
->get_selected_indices();

 for my $selection (reverse sort @selected) {
 unless ($selection == -1) {
  splice(@{$STORE{'GLADEXML'}->get_widget('ppdTrwConsole')-> {'data'}}, $selection, 1);
 }
 }
 return TRUE;
}
```

Así mismo, asociado al componente “ppdCbeConsole”, que muestra la lista de consolas para seleccionar una de ellas, se le asigna un manejador de señal, llamado “on_ppdCbeConsole_changed”, que responde cuando se produce algún cambio en este widget, es decir, cuando se selecciona un elemento consola de la lista.

En este método, se incluye la posibilidad de poder escribir en el campo de texto relativo al contenido de la etiqueta <xterm>. Cuando el elemento seleccionado en el widget “ppdCbeConsole” es “xterm”, entonces el widget “ppdEntRtXterm” aparecerá habilitado para escribir, en caso contrario se mostrará deshabilitado.

Ésto se resume en el siguiente fragmento de código:

```
sub on_ppdCbeConsole_changed() {
 my ($widget, $event) = @_;
 &debug("$widget received $event");

 if ($STORE{'GLADEXML'}->get_widget('ppdCbeConsole')->child()->get_text() eq 'xterm') {
 $STORE{'GLADEXML'}->get_widget('ppdEntRtXterm')->set_sensitive(TRUE);
 } else {
 $STORE{'GLADEXML'}->get_widget('ppdEntRtXterm')->set_sensitive(FALSE);
 }
}
```

A parte de estas funciones que responden a eventos, hacen falta incluir algunas otras instrucciones en otros métodos del código de VNUMLGUI para que queramos conseguir el resultado buscado.

Uno de los objetivos que buscamos es que una vez q el usuario ha introducido datos en los campos de esta pestaña Consoles y ha cerrado la ventana Project Properties mediante el botón “Aceptar”, aparezcan todos los valores introducidos por el usuario en la sección xml del programa VNUMLGUI. Ésto se consigue implementandolo en el método de volcado del xml, “dump_xml()” y en el método “apply_properties_dialog()”, en donde la variable STORE almacena los valores introducidos por el usuario.

Antes de nada, hay que matizar que el contenido correspondiente a <console> se almacena en la estructura STORE con la siguiente forma:

STORE{'SIMULATION'}{'CONSOLE'} es un array en el que cada elemento es un hash formado por dos claves: console y console_id.

STORE{'SIMULATION'}{'CONSOLE'}= [[(console => '.....', console_id => '.....')], ...].

```
sub apply_properties_dialog() {
....
....
# console
@{$STORE{'SIMULATION'}{'CONSOLE'}} = ();
 my $id=-1;
 foreach my $console (@{$STORE{'GLADEXML'}->get_widget('ppdTrwConsole')->{'data'}}) {
 $id++;
 push(@{$STORE{'SIMULATION'}{'CONSOLE'}}, {'console'=> $console->[0], 'console_id'=>$id});
 }
 if ($id>-1){
        $flag_vm_defaults = TRUE;
 }

 # xterm
 if ($STORE{'GLADEXML'}->get_widget('ppdCbeConsole')->child()->get_text() eq 'xterm') {
        if ($STORE{'GLADEXML'}->get_widget('ppdEntRtXterm')->get_text() ne ''){
        $STORE{'SIMULATION'}{'XTERM'} = $STORE{'GLADEXML'}->
get_widget('ppdEntRtXterm')->get_text();
        }else{
                delete($STORE{'SIMULATION'}{'XTERM'});
        }
 }
.....
```

```
.....
}


sub dump_xml() {
.....
.....
 # console
  if (defined ($STORE{'SIMULATION'}{'CONSOLE'})){
          my $flag = 0;
          foreach my $console (@{$STORE{'SIMULATION'}{'CONSOLE'}}) {
                  $text.= "\t\t\t<console";
                  $text.= " id=\"".$console->{'console_id'}. "\"";
                  $text.= ">". $console->{'console'}. "</console>\n";
                  $flag = 1 if ($console->{'console'} eq 'xterm');
          }
  # xterm
  $text.= "\t\t\t<xterm>". $STORE{'SIMULATION'}{'XTERM'}. "</xterm>\n" if
(defined($STORE{'SIMULATION'}{'XTERM'}) && $flag);
  }
.....
.....
}
```

Otro de los objetivos es, si se volviese a abrir de nuevo la ventana Project Properties del menu Archivo, deberían aparecer los datos introducidos anteriormente. Es decir, se pretende que las consolas que se habían ido añadiendo en el árbol, se muestren adecuadamente y en el mismo orden, y el campo de texto correspondiente a la etiqueta <xterm> aparezca habilitado si el elemento en la entrada del ComboBox de consolas se corresponde con "xterm", en caso contrario, este campo de texto aparecería deshabilitado.

Ya que la etiqueta <xterm> tiene la propiedad de ser única, es decir sólo puede aparecer una vez en la especificación xml, se pretende también que el valor de esta etiqueta aparezca escrito en su campo de texto cada vez que se abra la ventana Project Properties.

Todo esto se implementa en el método "setup_properties_dialog()". Este método rellena los campos de la ventana Project Properties con los valores almacenados en la estructura STORE.

```
sub setup_properties_dialog() {
....
....
# console

 my $con = Gtk2::SimpleList->new_from_treeview($STORE{'GLADEXML'}->
get_widget('ppdTrwConsole'), 'Console' => 'text');
 # we clear the Cbe
 $STORE{'GLADEXML'}->get_widget('ppdCbeConsole')->get_model()->clear();
 # we clear the Ent
 $STORE{'GLADEXML'}->get_widget('ppdEntRtXterm')->set_text('');

 foreach my $ex (qw/xterm pts/) {
 # fill the Cbe with possible console values
 $STORE{'GLADEXML'}->get_widget('ppdCbeConsole')->append_text($ex);
 }
```

```
 foreach my $console (@{$STORE{'SIMULATION'}{'CONSOLE'}}) {
 &debug("adding console in Trw");
 push(@{$STORE{'GLADEXML'}->get_widget('ppdTrwConsole')->{'data'}}, $console->{'console'});
 }

#xterm
 if ($STORE{'GLADEXML'}->get_widget('ppdCbeConsole')->child()->get_text() eq 'xterm'){
        $STORE{'GLADEXML'}->get_widget('ppdEntRtXterm')->set_sensitive(TRUE);
 }else {
        $STORE{'GLADEXML'}->get_widget('ppdEntRtXterm')->set_sensitive(FALSE);
 }

 if ($STORE{'SIMULATION'}{'XTERM'}) {
  $STORE{'GLADEXML'}->get_widget('ppdEntRtXterm')->
set_text($STORE{'SIMULATION'}{'XTERM'});
 }
....
....
}
```

Cuando se abre una simulación existente definida en un fichero xml, se pretende que, igual que en los casos anteriores, se rellenen los campos correspondientes con los valores especificados en el xml. Para ello la variable STORE almacena los valores definidos en la especificación del xml, mediante el siguiente fragmento de código en la función open_file().

```
sub open_file() {
....
....
# console
 my $console_list = $vm_defaults->item(0)->getElementsByTagName("console");
 for (my $i = 0; $i < $console_list->getLength; $i++) {
  my $console = $console_list->item($i);
  push(@{$STORE{'SIMULATION'}{'CONSOLE'}}, { 'console' => &text_tag($console), 'console_id'
=> $console->getAttribute("id")});
 }

 # xterm
 my $xterm = $vm_defaults->item(0)->getElementsByTagName("xterm"); $dh->get_default_xterm();
 if ($xterm->getLength > 0) {
        $STORE{'SIMULATION'}{'XTERM'} = $dh->get_default_xterm();
 }
....
....
}
```

Como hemos comprobado, el introducir modificaciones en la DTD del paquete VNUML, conlleva diferentes modificaciones e implementaciones en los métodos del código de VNUMLGUI.

Para este ejemplo concreto que acabamos de describir, volvamos a resumir los pasos que se siguen para que todo funcione correctamente:

1. Se parte de una modificación concreta en la DTD realizada por el equipo de VNUML:

   La creación de dos nuevas subetiquetas, <console> y <xterm>, hijas de la etiqueta <vm_defaults>, a su vez subetiqueta de <global>.

2. La introducción de estas dos etiquetas producen cambios en la apariencia de la ventana Project Properties. Estos cambios se reflejan en las siguientes subrutinas del código:

    - Subrutinas de señal que responden a los eventos de nuevos widgets (en este caso concreto se crean 3 nuevas subrutinas).

    - Modificaciones en la subrutina “apply_properties_dialog()”.

    - Modificaciones en la subrutina “setup_properties_dialog()”.

    - Modificaciones en la subrutina “dump_xml()”.

    - Modificaciones en la subrutina “open_file()”.

Los pasos anteriormente seguidos nos sirven de orientación del trabajo desarrollado que se produce para cada una de las actualizaciones de la DTD. Para las demás actualizaciones, se producirán modificaciones en el código de VNUMLGUI similares a los descritos, particularizando en cada caso lo que sea necesario.

En los casos en los que se ha tenido que implementar la apariencia relativa a las etiquetas que figuran en alguno de los cuadros de diálogo de Virtual Machine Properties, Net Properties y Host Properties, volvemos a escribir el paso 2 anterior ya que varía el nombre de las funciones que son modificadas:

2. La introducción de etiquetas que producen cambios en la apariencia de la ventanas Virtual Machine Properties, Net Properties y/o Host Properties, reflejan modificaciones en las siguientes subrutinas del código:

    - Creación y/o modificación de las subrutinas de señal que responden a los eventos de nuevos widgets.

    - Modificaciones en la subrutina “dialog_run()”.

    - Modificaciones en la subrutina “dialog_reset()”.

    - Modificaciones en la subrutina “dump_xml()”.

    - Modificaciones en la subrutina “open_file()”.

    - Modificaciones en la subrutina “new()” del package Routes, Switch y/o Host, modificando y/o introduciendo los campos necesarios en las siguientes estructuras:

```
my %router_fields = (
    'order'         => 1,
    'kernel'        => 1,
    'memory'    => 1,
    'filesystem' => 1,
    'filesystem_type'  => 1,
    'interfaces'  => 1,
    'routes'        => 1,
    'filetrees'     => 1,
    'ip_forward'  => 1,
    'disable_mng_if'  => 1,
    'execs'   => 1,
```

```
        'consoles'  => 1,
        'users'     => 1,
        'shell'     => 1,
        'basedir'   => 1,
        'xterm'     => 1,
        );
    my %switch_fields = (
        'mode'     => 1,
        'type'     => 1,
        'external' => 1,
        'hub'      => 1,
        'vlan'     => 1,
        'bandwidth' => 1,
        'uml_switch_binary' => 1,
        'capture_file'  => 1,
        'capture_expression' => 1,
        'scope'    => 1,
        'sock'     => 1,
        'capture_dev'  => 1,
        'x'        => 1,
        'y'        => 1,
        );
    my %host_fields = (
        'physical_interfaces' => 1,
        'interfaces'    => 1,
        'routes'        => 1,
        'ip_forward'    => 1,
        'execs'         => 1,
        );
```

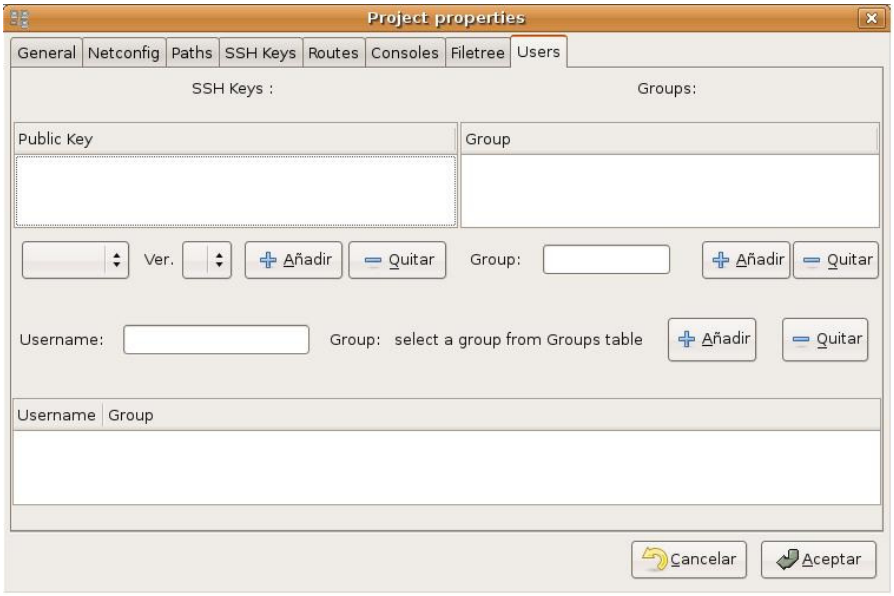
Si nos referimos a los casos de las etiquetas que aparecen en la ventana Preferencias del menu Edit, entonces el punto 2 anterior se modifica de la siguiente manera:

2. La introducción de etiquetas que producen cambios en la apariencia de la ventana Preferences, reflejan modificaciones en las siguientes subrutinas del código:

- Creación y/o modificación de las subrutinas de señal que responden a los eventos de nuevos widgets.
- Modificaciones en la subrutina “apply_preferences_dialog()”.
- Modificaciones en la subrutina “setup_preferences_dialog()”.
- Modificaciones en la subrutina “dump_xml()”.
- Modificaciones en la subrutina “open_file()”.

A continuación se explica brevemente las estructuras definidas y el funcionamiento del interfaz relativo a la etiqueta <user>. Ya que anteriormente se han descrito técnicamente los pasos para cronstruir una parte del interfaz gráfico y los cambios introducidos en el código, en este caso nos vamos a centrar en los objetos y estructuras en código perl que se han creado para esta etiqueta.

Como ya habíamos mostrado anteriormente, el interfaz que se ha originado para la etiqueta <user>, es el siguiente:



A partir de la imagen se puede observar, que en esta pestaña existen 3 árboles, uno para los usuarios que se van añadiendo (ya que la etiqueta <user> puede aparecer una o más veces), y los otros dos para sus subetiquetas <group> y <ssh_key>, las dos también con la propiedad de multiplicidad.

Para introducir un nuevo usuario se rellena el campo Username y a continuación se elige un grupo de la lista Groups haciendo doble click sobre él. La etiqueta "select a group from Groups table" se sustituye por el nombre del grupo que se haya seleccionado. Si no se rellena el campo Username o no se selecciona ningún grupo de la lista, no se podrá añadir el nuevo usuario, ésto se indica con un cuadro de diálogo de error. Tampoco se podrá introducir un nuevo usuario con el mismo Username de alguno de los que ya han sido añadidos anteriormente, ésto también se indica on otro cuadro de diálogo.

Una vez hayamos introducido varios usuarios en la lista de usuarios, se pueden ver y/o editar las propiedades de cada usuario (lista de groups, lista de ssh_keys), haciendo doble click sobre la fila del usuario en cuestión.

Debido al hecho de que la variable global STORE o las demás estructuras que se utilizan para guardar los valores de la especificación, no almacenan los datos hasta que no se acciona el botón "Aceptar" de la ventana, para poder mostrar las propiedades de los usuarios introducidos en la lista, ha sido necesario crear una estructura auxiliar que fuese almacenando los datos de la configuración del interfaz Users, para poder acceder a ellos mientras se van creando nuevos usuarios y todavía no se ha pulsado el botón Aceptar.

La variable que se origina es el hash array %UserHash. Cada elemento de esta estructura tiene la siguiente forma:

```
%UserHash{'USER'} = {'username' ->....,
                     'group' ->....,
                     'usergroup' => [@usergroup_list],
                     'usersshkey'=> [@userkeys_list]
                     }
```

Las dos últimas claves, “usergroup” y “usersshkey”, relativas a las dos subetiquetas de <user>, tienen cada una como valor una lista asociada cuyos elementos pertenecen a las listas SSH_keys y Groups respectivamente.

Una vez que se acciona el botón “Aceptar” de la ventana en la que nos encontramos, si en la estructura UserHash se han almacenado datos entonces éstos se copian al hash STORE, si estamos tratando la etiqueta <user> hija de <vm_defaults> (ventana Project Properties). Si lo que se está modificando es el contenido de <user> subetiqueta de <vm> (ventana Virtual Machine Properties), se copian a la estructura encargada de almacenar los datos de la configuración del paquete Route. Una vez que estos datos son copiados a estas estructuras, se podría realizar el volcado del xml adecuadamente.

Para editar las propiedades de un usuario existente, se pulsa doble click sobre el usuario en cuestión, el programa responde activando un flag, $user_activated, para indicar que los datos de este usuario han de mostrarse en cada una de las listas del interfaz Users, y almacenando en la variable $user_item, la posición del usuario correspondiente del hash array UserHash, para realizar posteriormente las modificaciones pertinentes sobre el usuario de esa posición del array.

El código siguiente muestra dos funciones importantes en este interfaz Users, el método encargado de insertar un nuevo usuario, “on_ppdButUserAdd_clicked()”, y el método que responde al evento de seleccionar un usuario existente para ver/editar sus propiedades, “on_ppdTrwUsers_row_activated()”:

```
sub on_ppdButUserAdd_clicked() {
 my ($widget, $event) = @_;
 &debug("$widget received $event");

 if ($STORE{'GLADEXML'}->get_widget('ppdEntUsername')->get_text() eq ''){
        &dialog('error', gettext("You must enter a username"));
        return FALSE;
 } else {
        my $size=$#{$STORE{'GLADEXML'}->get_widget('ppdTrwUsers')->{'data'}};
        my $item = -1;
        my $username = '';
        while ( $item < $size && $username ne $STORE{'GLADEXML'}->
get_widget('ppdEntUsername')->get_text()){
                        $item++;
                        $username = $STORE{'GLADEXML'}->get_widget('ppdTrwUsers')->
{'data'}[$item]->[0];
                }
        if ($username eq $STORE{'GLADEXML'}->get_widget('ppdEntUsername')->get_text()){
        &dialog('error', gettext("This user already exists. You must enter other different username."));
                return FALSE;
        } else {

                if (($STORE{'GLADEXML'}->get_widget('ppdLblUserGroup')->get_text() eq '') ||
($STORE{'GLADEXML'}->get_widget('ppdLblUserGroup')->get_text() eq 'select a group from Groups
table')) {
                &dialog('error', gettext("You must select a group from Groups table"));
                return FALSE;
                } else {
                push(@{$STORE{'GLADEXML'}->get_widget('ppdTrwUsers')->{'data'}},
                [ $STORE{'GLADEXML'}->get_widget('ppdEntUsername')->get_text(),
$STORE{'GLADEXML'}->get_widget('ppdLblUserGroup')->get_text()]);
```

```
                my @usergroup_list=();
                foreach my $usergroup (@{$STORE{'GLADEXML'}-
>get_widget('ppdTrwUserGroups')->{'data'}}) {
                push (@usergroup_list, $usergroup->[0] );
                }

                my @userkeys_list=();
                foreach my $userkeys (@{$STORE{'GLADEXML'}->get_widget('ppdTrwUserKeys')->
{'data'}}) {
                push (@userkeys_list, $userkeys->[0] );
                }

                push (@{$UserHash{'USER'}}, {'username' => $STORE{'GLADEXML'}->
get_widget('ppdEntUsername')->get_text(), 'group' => $STORE{'GLADEXML'}->
get_widget('ppdLblUserGroup')->get_text(), 'usergroup' => [@usergroup_list], 'usersshkey' =>
[@userkeys_list]});

                $STORE{'GLADEXML'}->get_widget('ppdTrwUserGroups')->get_model()->clear();
                $STORE{'GLADEXML'}->get_widget('ppdTrwUserKeys')->get_model()->clear();
                $STORE{'GLADEXML'}->get_widget('ppdEntUsername')->set_text('');
$STORE{'GLADEXML'}->get_widget('ppdEntUserGroup')->set_text(''); $STORE{'GLADEXML'}-
>get_widget('ppdLblUserGroup')->set_text('select a group from Groups table');

                return TRUE;
                }
                }
        }
}

sub on_ppdTrwUsers_row_activated() {
 my ($widget, $event) = @_;
 &debug("$widget received $event");

 my @selected = $STORE{'GLADEXML'}->get_widget('ppdTrwUsers')->get_selected_indices();
 my $username = '';
 for my $selection (reverse sort @selected) {
 unless ($selection == -1) {
  $username = $STORE{'GLADEXML'}->get_widget('ppdTrwUsers')->{'data'}[$selection]->[0];
  }
 }
 $STORE{'GLADEXML'}->get_widget('ppdTrwUserGroups')->get_model()->clear();
 $STORE{'GLADEXML'}->get_widget('ppdTrwUserKeys')->get_model()->clear();
 #my $select = '';
 my $item = 0;
 if ($username ne ''){
        while ($UserHash{'USER'}[$item]->{'username'} ne $username){
                $item++;
        }
        $user_item = $item;
        foreach my $group (@{$UserHash{'USER'}[$item]->{'usergroup'}}) {
        push(@{$STORE{'GLADEXML'}->get_widget('ppdTrwUserGroups')->{'data'}}, $group);
        }
        foreach my $key (@{$UserHash{'USER'}[$item]->{'usersshkey'}}) {
          push (@{$STORE{'GLADEXML'}->get_widget('ppdTrwUserKeys')->{'data'}}, $key);
        }
        $user_activated = TRUE;
 }
```

```
 return TRUE;
}
```

3.2 Detección del tráfico de paquetes

Debido al propósito didáctico de VNUML y de VNUMLGUI, hemos considerado de gran interés el desarrollo de un analizador con fines docentes, de uso fácil y libre distribución adaptado a la interfaz gráfica de VNUMLGUI. Este analizador de protocolos sería capaz de monitorizar las redes virtualizadas.

Por uso fácil queremos decir que el programa realiza exclusivamente las tareas que hemos considerado, son las requeridas por sus usuarios. Es decir, las opciones de configuración del programa son mínimas y no se requiere que el usuario posea conocimientos acerca de las opciones posibles en un analizador comercial.

Así pues, a modo de resumen las características de nuestro analizador han de cumplir los siguientes requisitos:

- integrable con VNUMLGUI, y por tanto ha de correr bajo Linux, bajo licencia GPL.
- capaz de capturar tramas de interfaces virtuales.
- que sea mostrado en tiempo real.
- de fácil uso.

3.2.1 Drivers TUN / TAP

Para conseguir la comunicación de aplicaciones externas al núcleo, como VNUML, con el espacio del kernel existen una serie de subrutinas o funciones que permiten al programador la interacción de ambos espacios. Nos centraremos en estudiar los dispositivos usados por VNUML para conseguir este propósito. VNUML soluciona este problema con el uso de dispositivos TAP, implementando un dispositivo TAP para permitir mostrar capturas en tiempo real. Veamos que son estos dispositivos.

Los dispositivos TUN/TAP son interfaces virtuales de red (ubicadas en /dev/net/), diferentes de las clásicas interfaces físicas (eth0) las cuáles necesitan de una tarjeta de red y cables. Son usadas para proveer recepción y transmisión de paquetes para programas del espacio-de-usuario. Pueden ser vistos como un simple dispositivo Punto a Punto o Ethernet, el cuál en vez de recibir paquetes a través de un medio físico, los recibe desde programas del espacio-de-usuario y en vez de enviarlos por un medio físico los escribe en el espacio de usuario.

Así, una aplicación creará un fichero de dispositivo /dev/net/tap y con eso creará una nueva interfaz de red (por defecto tap0). Todo lo que la aplicación escriba en ese dispositivo se recibirá en la interfaz de red recién creada; de igual modo todo lo que llegue a esa interfaz de red (por ejemplo a través de enrutamiento) lo leerá la aplicación del fichero de dispositivo.

Los dispositivos TUN/TAP están muy relacionados. Veamos en que difieren, y por qué VNUML utiliza el dispositivo de tipo TAP.

TUN es un dispositivo Punto a Punto virtual, operando en la capa 3 capturando tramas IP punto a punto. El driver TUN fue desarrollado como soporte de bajo nivel para el kernel para hacer IP tunneling. TUN provee a las aplicaciones de espacio-de-usuario 2 interfaces:

- /dev/tunX - un dispositivo de carácter

- tunX - una interfaz Punto a Punto virtual.

Las aplicaciones de espacio-de-usuario pueden escribir tramas IP a /dev/tunX y el kernel las recibe a través de la interfaz tunX. Al mismo tiempo, cada trama que el kernel escribe en la interfaz tunX, podrá ser leída por los programas de espacio-de-usuario o a través de la interfaz /dev/tunX.

TAP es un dispositivo Ethernet virtual. Opera pues, en la capa 2 capturando tramas Ethernet y por tanto multipunto. El driver TAP fue desarrollado como soporte de bajo nivel para el kernel para Ethernet tunneling. Igualmente provee a las aplicaciones del espacio-de-usuario 2 interfaces:

- /dev/tapX - un dispositivo de carácter

- tapX - una interfaz Ethernet virtual

Los programas del espacio-de-usuario pueden escribir las tramas Ethernet en /dev/tapX y el kernel las recibirá a través de la interfaz tapX. De la misma forma cuando el kernel escribe en tapX, los programas de espacio-de-usuario recibirán las tramas por /dev/tapX.

En nuestro caso particular, se establecen los flags del dispositivo indicándole que la interfaz tiene que funcionar en modo promiscuo, lo cual significa que recibirá y procesará todos los paquetes que escuche, aunque en principio no vayan dirigidos a ésta interfaz.

Una vez que hemos explicado cómo se crean los dispositivo TAP, y sin querernos centrar en como crear los paquetes ya que no nos concierne, sólo debemos estar atentos al tráfico que circula por esas interfaces, y en cómo decodificarlo.

Todas estas cuestiones requieren tener cierto conocimiento de detalles de linux, que el usuario no debe por que tener. Nuestro objetivo ha sido intenta hacer lo máximo posible para que estos procesos sean transparentes al usuario y que este únicamente se centre en definir el sistema emulado.

Con esto ya tenemos la pieza clave para la captura de paquetes en tiempo real. Pero no sólo eso. Estos paquetes serán decodificados paralelamente, y nuestro objetivo será mostrarlos al usuario final de forma asíncrona, para que no actúe en contra del rendimiento.

3.3 Decodificación de paquetes

Una vez creada las interfaces virtuales, lo primero que tenemos que hacer es instanciarlas. Esto lo hacemos mediante la librería Net::Pcap que provee una interfaz para la captura de paquetes a nivel usuario. Esta biblioteca proporciona un marco portable para monitorización y es capaz de crear estadísticas de la red, supervisar la seguridad, y eliminar errores de la red. Nos centramos en ella, explicando las principales funciones usadas.

A continuación describiremos los pasos necesarios para dicha instanciación (lo cual se encuentra en el método create_pcap())

El primer paso en el uso de esta herramienta es determinar cuáles interfaces de la red vamos a querer monitorizar. Estos dispositivos se pueden especificar, o se pueden determinar por el método del lookupdev. Veamos las dos posibilidades:

La sintaxis del método del lookupdev es como sigue:

$dev = Net::Pcap::lookupdev(\$err)

Este método devuelve el nombre del primer dispositivo de red que se pueda utilizar para supervisar tráfico de la red.

Un segundo método del paquete que autoriza la introducción del nombre de la interfaz de red es "lookupnet", que se puede utilizar para determinar la dirección de red y la máscara de red para un dispositivo. Este método es útil para la validación de un nombre de dispositivo proveído para la red por un usuario. La sintaxis del método del lookupnet es como sigue:

$result = Net::Pcap::lookupnet($dev, \$net, \$mask, \$err);

Este método devuelve la dirección y la máscara de red para el dispositivo especificado: $dev. Devuelve 0 si tiene éxito y -1 si falla

Con los argumentos resueltos de este método, una vez conocidas la dirección, y la máscara de red, el segundo paso es obtener un descriptor de fichero, para ello usamos la función open_live (también de la librería Pcap) que devuelve un descriptor de paquetes que posteriormente se puede utilizar para capturar y examinar los paquetes de la red. La sintaxis del método open_live es como sigue:

$pcap_t = Net::Pcap::open_live($dev, $snaplen, $promisc, $to_ms, \$err)

$dev especifica el interfaz de la red de el cual capturar. Los parámetros de $snaplen y de $promisc especifican el número máximo de bytes de cada paquete y si poner el interfaz en módo promiscuo (dónde los paquetes de la red dirigidos no necesariamente a la máquina que captura el paquete se capturan), respectivamente. El parámetro de $to_ms especifica un time-out en milisegundos. Con esta función obtenemos la estructura pcap_t necesaria para proceder a analizar los paquetes.

Esta función la utilizamos en modo NO promiscuo, solo queremos los paquetes que realmente vayan dirigidos a la interfaz en cuestión.

Hasta aquí los métodos descritos arriba proporcionan los medios para capturar todo el tráfico de la red, ahora pasaremos a describir los métodos que son capaces de seleccionar paquetes para supervisar un tráfico específico. La filtración usa un lenguaje específico, usando el estándar de filtros que se puede encontrar en el código fuente del libpcap o en las páginas del manual de tcpdump y del cual no cometaremos nada más.

Veamos los métodos proporcionados por Net::Pcap para la compilación y filtrado de la captura de paquetes: Net::Pcap::compile y Net::Pcap::setfilter.

Net::Pcap::compile($pcap_t, \$filter_compiled, $filter_string, $optimise, $netmask)

Este método compilará y comprobará el filtro especificado en $filter_string para $pcap_t y devuelve el filtro compilado en el escalar $filter_compiled. El filtro se optimiza si la variable $optimise es verdadera. Esta función, devuelve 0 si es aceptado o -1 si ocurre un error.

Para aplicar el filtro se usa la función set_filter aplicándolo a $pcap_t.

Net::Pcap::setfilter($pcap_t, $filter_compiled);

Sobre los filtros se hablara de nuevo en las conlusiones de este apartado.

El paso siguiente en el proceso de la captura del paquete es establecer una función de callback a la cual se le pasan los paquetes capturados para el análisis. Para esto, se usa la función de GLib::IO::add_watch() que llama de forma concurrente a un método que decodifica los paquetes. Activándose cada vez que se recibe un paquete. Sus características y uso, están definidos un poco más adelante, en el apartado 3.4: "Asincronismo VNUMLGUI".

```
my $watch=Glib::IO->add_watch($file, ['in', 'hup'], sub {…}
```

Una vez acabada la captura de los paquetes, el método "close" de Net::Packet se debe llamar para cerrar el dispositivo de captura del paquete. Para llamarle:

Net::Pcap::close ($pcap_t)

Por si acaso también se borra el add_watch() con el método de la clase Glib::Source

Glib::Source->remove($add_Watch)

Además se deben borrar las variables almacenadas de esas instanciaciones que se encuentran en la estructura correspondiente de la net dentro de su campo watch.

3.3.1 Módulos NetPacket

Hasta ahora hemos visto que los paquetes que vamos a recibir van a ser de la capa 2. La capa 2 utiliza la difusión para su transmisión de paquetes, de modo que cualquier tarjeta conectada a una red, recibirá el paquete, sea o no sea para él. Sin embargo, sólo la tarjeta con la dirección Ethernet que coincide con la Dirección indicada en la trama la aceptará, el resto simplemente la ignorará. Para la transmisión de paquetes se utilizan las Trama Ethernet; ésta vendría a ser el medio de transporte de todos los protocolos que se definen en las capas superiores. Las tramas son el formato en que los datos son encapsulados para poder ser transmitidos al medio físico. Se puede ver su estructura en el fichero ethernet.h en /usr/includes/net).

Una vez que se conocen los campos de una trama Ethernet cuando un paquete es recibido, se puede retirar la cabecera de Ethernet(14 bytes) y el checksum de verificación de la trama, comprobar que los datos corresponden a un mensaje IP y entonces pasarlo a dicho protocolo para que lo procese. Entonces se hará hincapié en el campo protocol o siguiente cabecera (si se está ante una trama ipv6) ya que servirá para determinar cual será la tercera y última cabecera a extraer. Existen multitud de protocolos, los más importantes podrían ser ICMP, TCP, UDP, IPV6.

Ahora se pasará a ver como conseguir decodificar más concretamente un paquete, y que módulos se usa para lo anteriormente explicado.

Así pues, una vez que se hayan capturado los paquetes, el paso siguiente es decodificar los paquetes y dar sentido a los datos del paquete de la red recogidos. Esto se puede realizar, construyendo plantillas para los datos capturados o más fácilmente con la colección de módulos NetPacket. Esta última forma es la que se ha utilizado, por ser claramente más simple. Estos módulos contienen los métodos para extraer la información de los paquetes de la red, el más útil en nuestro caso es el método de "decode": Este método devuelve una tabla hash de meta datos, la cuál es específica para cada tipo del paquete.

Así pues, se centrará en la colección de módulos de NetPacket.

NetPacket es un conocido framework para enviar y recibir tramas desde los capas 2 a 7 de una forma fácil. Con la nueva implementación de NetPacket se dá paso a una interfaz simple, y a un poderoso motor para manejo de las capas. Cada módulo del descendiente de NetPacket sabe codificar y descifrar los paquetes para el protocolo de red del que trata. Así, con NetPacket es posible encapsular cualquier cosa en cualquier cosa (por ejemplo NetPacket::VLAN), mandar el frame fácilmente y capturar la respuesta automáticamente. Los protocolos que NetPacket puede codificar/descifrar incluyen IPv4, el TCP, el UDP, el ICMP, Ethernet, y el ARP.

Existen pues multitud de paquetes NetPacket; se hablará principalmente en los siguientes módulos, los usados en nuestro código:

- NetPacket::Ethernet -> codifica y decodifica paquetes Ethernet
- NetPacket::IP-> codifica y decodifica tramas IP
- NetPacket::ARP-> codifica y decodifica tramas ARP
- NetPacket::ICMP-> codifica y decodifica tramas ICMP
- NetPacket::UDP-> codifica y decodifica tramas UDP
- NetPacket::UTP-> codifica y decodifica tramas UTP

Además NetPacket implementa NetPacket::IGMP (todos ellos se encuentran disponibles en www.cpan.org.)

3.3.1.1 Ethernet

Nos centraremos primero en el NetPacket::Ethernet, las funciones que implementa este paquete, son las siguientes:

```
$eth_obj = NetPacket::Ethernet->decode($raw_pkt);
$eth_data = NetPacket::Ethernet::strip($raw_pkt);
```

Vemos lo que hacen más detenidamente:

```
$eth_obj = NetPacket::Ethernet->decode($raw_pkt);
```

Descifra los datos crudos del paquete Ethernet y devuelve un objeto que contiene datos del paquete Ethernet. Los datos del paquete $raw_packet han de ser válidos.

Nos devolverá la información siguiente sobre los paquetes Ethernet capturados:

- src_mac: dirección MAC de la fuente
- dest_mac: dirección MAC de destino
- tipo: el tipo del protocolo del paquete de Ethernet, por ejemplo ETH_TYPE_IP ETH_TYPE_ARP ETH_TYPE_APPLETALK ETH_TYPE_SNMP ETH_TYPE_IPv6 ETH_TYPE_PPP
- datos: la carga útil de los datos para el paquete de Ethernet

$eth_data = NetPacket::Ethernet::strip($raw_pkt);

Devuelve los datos encapsulados (o la carga útil) contenidos en el paquete Ethernet. Estos datos pueden ser convenientemente utilizados como entrada para otros módulos NetPacket::*.

Un ejemplo de acceso sería:

```
$eth_obj = NetPacket::Ethernet->decode($packet);
print("$eth_obj->{src_mac}:$eth_obj->{dest_mac} $eth_obj->{type}\n");
```

3.3.1.2 IP

Pasemos a describir las características de otro paquete NetPacket::IP, este paquete además implementa otra función adicional, aún no implementada en el anterior paquete:

```
$ip_obj = NetPacket::IP->decode($raw_pkt);
$ip_pkt = NetPacket::IP->encode($ip_obj);
$ip_data = NetPacket::IP::strip($raw_pkt);
```

Los métodos decode() y strip() hacen lo mismo que los anteriormente descritos pero relativos a paquetes IP, y veamos el nuevo método encode():

NetPacket:: IP->encode()

Devuelve un paquete IP codificado con los datos del caso especificados. Esto deducirá la longitud total del paquete automáticamente del campo length de la carga útil y también ajustará el checksum.

Veamos los campos con los que se define un paquete de este tipo:

- ver: El número de versión del IP de este paquete. Este sólo puede ser de tipo IP_VERSION_IPv4.

- hlen: La longitud de la cabecera IP de este paquete.
- flags: Los flags de la cabecera IP para este paquete.
- foffset: El offset para este paquete.
- tos: El tipo de servicio para este paquete IP.
- len: La longitud (incluyendo la cabecera) en bytes para este paquete.
- Id: El número de la identificación (secuencia) para este paquete del IP.
- Ttl: El Tiempo de vida de este paquete.
- proto: El número del protocolo IP para este paquet. Estos pueden ser: IP_PROTO_IP IP_PROTO_ICMP IP_PROTO_IGMP IP_PROTO_IPIP IP_PROTO_TCP IP_PROTO_UDP
- cksum: El valor del checksum de paquete IP.
- src_ip: La dirección IP de la fuente para este paquete.
- dest_ip: La dirección IP del destino para este paquete.
- opciones: Cuales quiera opciones del paquete IP.
- Datos: Los datos encapsulados (carga útil) para este paquete del IP.

Su uso sería así:

```
$eth_obj = NetPacket::Ethernet->decode($packet);
if ($eth_obj->{type} eq ETH_TYPE_IP) {
$ip_obj = NetPacket::IP->decode($eth_obj->{data});
print("$ip_obj->{src_ip}:$ip_obj->{dest_ip} $ip_obj->{proto}\n");
…
}
```

3.3.1.3 ARP (y RARP)

Pasaremos a ver ahora el paquete NetPacket::ARP. Este módulo sólo implementa la siguiente función ya anteriormente explicada, pero para paquetes ARP:

```
$tcp_obj = NetPacket::ARP->decode($raw_pkt);
```

y sus objetos tendrán los siguientes campos:

- htype: Tipo del hardware.
- proto: Tipo del protocolo.
- hlen: Longitud de la cabecera.

- plen: Longitud del protocolo.
- opcode: Una de las constantes siguientes: ARP_OPCODE_REQUEST, ARP_OPCODE_REPLY, RARP_OPCODE_REQUEST, RARP_OPCODE_REPLY
- sha: Dirección del hardware de la fuente
- spa: balneario Dirección del protocolo de la fuente.
- tha: Dirección del hardware de la blanco.
- tpa: Dirección del protocolo de la blanco.

  Un ejemplo de uso, sería:

```
my $eth_obj = NetPacket::Ethernet->decode($packet);
if ($eth_obj->{type} eq ETH_TYPE_ARP) {
 my $arp_obj = NetPacket::ARP->decode($eth_obj->{data}, $eth_obj);
 print("source hw addr=$arp_obj->{sha}, ".
 "dest hw addr=$arp_obj->{tha}\n");
 }
```

3.3.1.4 ICMP

El módulo NetPacket::ICMP tiene también implementados los 3 métodos:

```
$icmp_obj = NetPacket::ICMP->decode($raw_pkt);
$icmp_pkt = NetPacket::ICMP->encode();
$icmp_data = NetPacket::ICMP::strip($raw_pkt);
```

Los campos de sus paquetes son:

- Tipo: El tipo de mensaje del ICMP de este paquete. Pueden ser: ICMP_ECHOREPLY ICMP_UNREACH ICMP_SOURCEQUENCH ICMP_REDIRECT ICMP_ECHO ICMP_ROUTERADVERT ICMP_ROUTERSOLICIT ICMP_TIMXCEED ICMP_PARAMPROB ICMP_TSTAMP ICMP_TSTAMPREPLY ICMP_IREQ ICMP_IREQREPLY ICMP_MASREQ ICMP_MASKREPLY
- Código: El código del mensaje del ICMP de este paquete.
- Cksum: El checksum para este paquete.
- Datos: Los datos encapsulados (carga útil) para este paquete.

Su uso sería:

```
$ip_obj = NetPacket::IP->decode($eth_obj->{data});
$protoc = getprotobynumber($ip_obj->{proto});
        if ($protoc eq 'icmp') {
         # Dissecting ICMP.
         my $icmp_obj = NetPacket::ICMP->decode($ip_obj->{data});
```

```
        if ($icmp_obj->{type} eq ICMP_ECHO) {
                $info='Echo (ping) request';
        }
        if ($icmp_obj->{type} eq ICMP_ECHOREPLY) {
                $info='Echo (ping) reply';
        }
        if ($icmp_obj->{type} eq ICMP_UNREACH) {
                $info='Destination unreachable (Network unreachable)'
        }
}
```

3.3.1.5 UDP

El módulo NetPacket::UDP tiene implementados los 3 métodos:

```
$udp_obj = NetPacket::UDP->decode($raw_pkt);
$udp_pkt = NetPacket::UDP->encode($ip_obj);
$udp_data = NetPacket::UDP::strip($raw_pkt);
```

Y sus paquetes tienen los campos siguientes:

- src_port: El puerto del UDP de la fuente del datagrama.
- dest_port: El puerto del UDP del destino del datagrama.
- len: La longitud (incluyendo cabecera) en bytes del paquete.
- cksum: El checksum de este paquete.
- datos: Los datos encapsulados (carga útil) para este paquete.

Su uso sería por ejemplo:

```
my $eth_obj = NetPacket::Ethernet->decode($packet);
$ip_obj = NetPacket::IP->decode($packet);
if($ip_obj->{proto} == IP_PROTO_TCP) {
my $udp_obj = NetPacket::UDP->decode($ip_obj->{data});
        print("$ip_obj->{src_ip}:$udp_obj->{src_port} -> ",
 "$ip_obj->{dest_ip}:$udp_obj->{dest_port} ",
 "$udp_obj->{len}\n");
 }
}
```

3.3.1.6 TCP

Por último: el módulo NetPacket::TCP. Sus funciones implementadas son:

```
$tcp_obj = NetPacket::TCP->decode($raw_pkt);
$tcp_pkt = NetPacket::TCP->encode($ip_pkt);
$tcp_data = NetPacket::TCP::strip($raw_pkt);
```

Y sus paquetes contienen los campos siguientes:

- src_port: El puerto del TCP de la fuente del paquete
- dest_port: El puerto del TCP del destino del paquete
- seqnum: El número de serie del TCP para este paquete
- acknum: El numero de ack del TCP para este paquete
- hlen: La longitud de la cabecera para este paquete
- reservado: Los 6 bits de espacio reservado en la cabecera del paquete TCP
- flags: Contiene las flags del urg, del ack, del psh, del rst, del syn, de la aleta, del ece y del cwr para este paquete: FIN SYN RST PSH ACK URG ECE CWR
- winsize: El tamaño del marco del TCP para este paquete
- cksum: El checksum del TCP
- urg: El indicador de urgencia del TCP
- opciones: Cualquieres opciones del TCP para este paquete en forma binaria
- datos: Los datos encapsulados (carga útil) para este paquete

El acceso sería:

```
my $eth_obj = NetPacket::Ethernet->decode($packet);
$ip_obj = NetPacket::IP->decode($packet);
if($ip_obj->{proto} == IP_PROTO_TCP) {
my $tcp_obj = NetPacket::TCP->decode( ip_strip( eth_strip($pkt)));
}
```

3.3.1.7 IPv6

Pero aunque NetPacket implementa muchos protocolos no implementa IPv6. Y llegados a este punto y sabiendo la importancia que tiene este protocolo en este proyecto hemos tenido que usar las funciones pack() y unpack()

Pack()/Unpack():nos permiten leer y escribir buffers de datos de acuerdo con una plantilla. Esta plantilla nos permite indicar cosas específicas como el orden de los bytes o el tamaño de la palabra, o también usar los valores por defecto. Si miramos perldoc -f pack o consultamos la función pack en Programming Perl, veremos una tabla que lista todos los caracteres que pueden ir en la plantilla, junto con una descripción del tipo de dato que significan.

Mediante esta función somos capaces de detectar los siguientes parámetros:

my ($vTcFl, $pl, $nh, $hl, $sa, $da, $payload) = unpack('NnCCa16a16 a*', $eth_obj-> {data});

- $vTcFl = version + traffic Class + Flow Label
- $pl: carga útil o datos
- $nh: siguiente cabecera
- $hl: hop limit
- $sa: dirección fuente
- $da: dirección destino

Con esto ya podemos obtener el campo protocolo:

$protoc = getprotobynumber($nh);

y así poder seguir decodificando los paquetes encapsulados según el tipo de protocolo que sea.

La decodificación de todos estos paquetes se haya en el método display_sniff() del paquete Switch.

3.4 Descripción y uso del analizador

Para comprender mejor el objetivo conseguido, se muestra a continuación un ejemplo de un escenario de VNUML en el que se usa la implementación anterior (Este código se distribuye junto con el software VNUML 1.7.1.0, en la sección Contrib.), desde el principio de la ejecución se capturará el tráfico entre las dos máquinas virtuales simuladas, con el siguiente comportamiento:

• Se desea capturar todo el tráfico transmitido entre las dos máquinas en tiempo real.

• Adicionalmente, se desea capturar el tráfico con origen y destino en el puerto 25

(Tráfico relativo al protocolo SMTP) volcándolo en un fichero para su posterior estudio.

Para conseguir el mismo resultado, se podría haber procedido de la siguiente manera.

Se construye una topología cualquiera.

Y a continuación, se invocan los siguientes comandos:

• umlctl direct set-capture_file "/tmp/capture0.cap" –sinname tutorial_capture Net0, para establecer el fichero de captura.

• umlctl direct set-capture_dev "prueba" –sinname tutorial_capture Net0, para establecer el nombre de la interfaz donde capturar el tiempo real.

• umlctl direct set-capture__lter "port 25" –sinname tutorial_capture Net0, para establecer el filtro de captura que se aplicará al volcado de paquetes en el fichero antes definido.

• umlctl direct start-capture_file –sinname tutorial_capture Net0, para comenzar la captura en fichero.

• umlctl direct start-capture_dev –sinname tutorial_capture Net0, para comenzar con la captura en la interfaz.

O bien actuar del siguiente modo:

• Ejecutar el demonio umlswitchd

• Construimos el escenario mediante vnumlparser.

• Ejecutamos las instrucciones antes enumeradas, sustituyendo la opción direct por daemon.

Como vemos se trata de una ejecución un tanto confusa y compleja, que requiere el uso de programas externos para la decodificación de esos paquetes capturados. Vemos ahora lo que hace falta hacer para ejecutar la nueva implementación.

Para poder hacer uso de nuestro analizador de protocolos, es necesario primero la instalación de los paquetes: NetPacket::*

El tráfico generado sólo se puede ejecutar cuando la simulación está en ejecución, siempre y cuando hayamos rellenado el campo “capture_dev” de net tipo “uml_switch”. En el caso de ser “virtual_bridge” no hará falta.

Dependiendo de los que queramos será necesario hacer una cosa u otra. Distinguimos entre analizador de protocolos y el “coloreador”.Más adelante definimos ambos.

- Si queremos capturar TODO el tráfico en tiempo real debemos pulsar “Ver->sniff all nets”: de esta forma aparecerán en la parte de abajo una pestaña “Sniff”, con varias subpestañas, una por red capturada. Además se nos colorearán los routers y switches por los que circulen los paquetes.

- Si queremos capturar solo una red en especial: pulsaremos botón derecho sobre el switch: “Sniff…” de esta forma se nos mostrará una pestaña en la parte inferior de la pantalla con la información recogida.

- Si queremos “colorear” un solo router, entonces pulsaremos botón derecho sobre el router elegido:”Highlight”.

- Si deseamos capturar los paquetes en un fichero, entonces deberemos rellenar el campo “Capture file” de cada switch a capturar. Estos ficheros tienen una extensión.cap. Por ejemplo:” /tmp/capture0.cap”.

- Si deseamos aplicar un filtro al fichero almacenado, debemos rellenar el campo Capture Exp de cada switch a capturar. Para ello debemos conocer la sintaxis del estándar de filtros.

Pasemos a describir las características de nuestro “sniff”. Para describir nuestro Sniff, hemos de dividirlo primero en dos partes: el analizador de protocolos o ‘Sniff’ propiamente dicho, y el 'coloreador' de elementos.

El 'Sniff' o comúnmente llamado analizador de protocolos muestra una línea por cada trama capturada (el número de secuencia, el instante de captura, origen y destino, tamaño, y protocolo más alto de los detectados) que pertenezca a la interfaz existente elegida.

Por otra parte y debido a las características de nuestro virtualizador, además hemos creado otra forma más visual de mostrar las rutas que seguían las tramas a través de las distintas interfaces virtuales. Para ello hemos implementamos un coloreador que actúa cada vez que un elemento recibe un paquete. (Como veremos esto hace de VNUMLGUI una aplicación perfecta para fines didácticos en cuanto a redes y protocolos)

Características de nuestro 'sniffer':

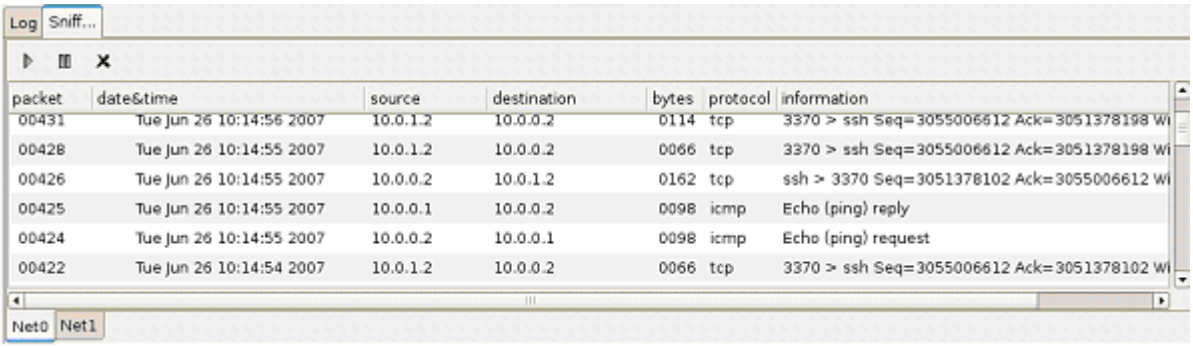
- ✓ Distribuido bajo licencia GPL, y corre bajo Linux.
- ✓ Trabaja en "no promiscious mode".
- ✓ Automatización de la creación de los dispositivos virtuales.
- ✓ Captura datos de interfaces virtuales. Es por ello que no necesita 'colocarse' correctamente, sino que es 'omnipresente'.
- ✓ Muestra los datos en la interfaz grafica, cómoda y fácil de leer.
- ✓ Almacena los datos en un fichero con formato estándar (capaz de ser usado por otros protocolos).
- ✓ Integrado con la interfaz gráfica VNUMLGUI
- ✓ Protocolos soportados: en principios todos. Aunque no todos muestran la misma cantidad de información.

Como ya hemos dicho anteriormente, nuestro analizador de protocolos tiene la característica de que es "omnipresente" que no hace falta colocar el analizador de protocolos en un lugar de la red en concreto, sino que éste accederá a cualquiera de las interfaces virtuales creadas.

Así pues una vez arrancado el servicio (sección anterior) podemos observar que aparecen en la zona inferior dos pestañas, empezando por la izquierda:

- La primera de ellas se corresponde con los 'logs'. En esta pestaña se muestran los Log tanto del programa VNUMLGUI, como del VNUMLparser. Podremos ver en tiempo real en que estado está la simulación, o si ha habido algún problema, a qué se debe éste.

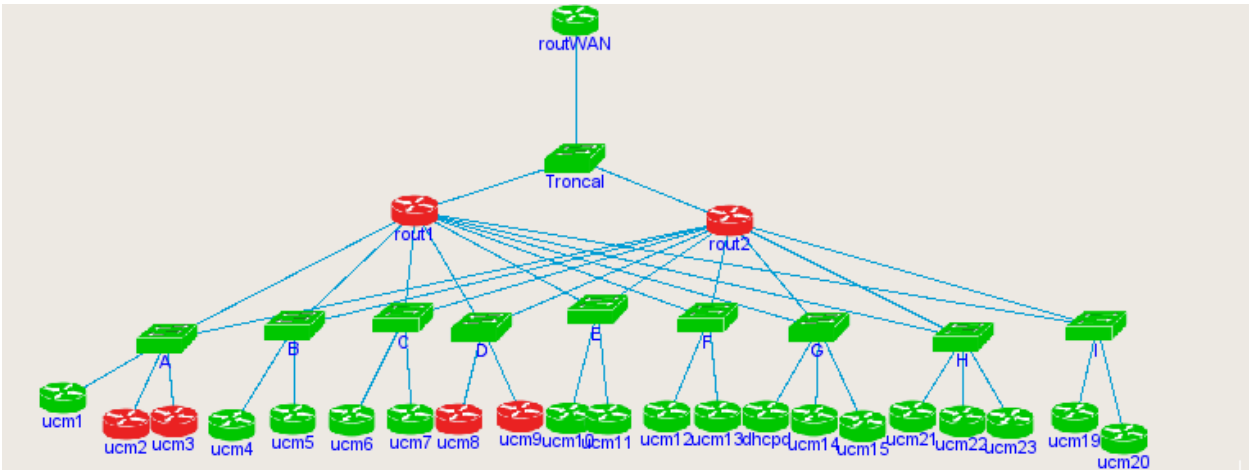
-La segunda de las pestaña se titula 'Sniff', y puede contener una o más pestañas (dependiendo del numero de interfaces arrancadas) cada una de las cuales se corresponde con cada una de las interfaces creadas en la simulación.



El "Coloreador" se activa cada vez que le llega un paquete ethernet a un switch o a un router o cuando ese mensaje es un mensaje de broadcast. Actúa cambiando de color tanto los Switches como los Routers, pero sólo cuando éstos han sido activados (de la forma anteriormente especificada):

```
if ( (grep { $_ =~ $source} @ifcmacs)
                ||(grep { $_ =~ $destin} @ifcmacs) ||
                 ($destin =~ /ffffffffffff/)  ||
                ($destin =~ /000000000000/ ) )
        {
        $self->set_color($red ? 'green': 'red');
        $red = 1 -$red;
        }
```

En la siguiente simulación hemos activado el coloreador de los routers, pero no el de las net. En este momento está circulando el tráfico entre ellos, y se encuentran en color rojo:



Conclusión

Nuestro sniffer no puede decodificar los datos de transporte, sólo los de red. Esto quiere decir, que puede detectar que es UDP, TCP, VRRP, ICMP… en los que se rellenará el campo Información, pero NO PUEDE DETECTAR QUE ES HTTP, BROWSER, etc, pudiendo ser factible su conclusión en un futuro trabajo de una manera facil y modular.

No nos ha sido posible integrar la filtración con el nuevo VNUMLGUI y con IPv6. Se deja abierto para nuevas implementaciones.

Las conclusiones que obtenemos de esta nueva implementación de este analizador de protocolos, es que hemos conseguido un "sniffer" mucho más accesible y practico, evitando la necesidad de utilización de otros programas de software. El complemento perfecto para un simulador de redes como VNUML/VNUMLGUI. Además sus objetivos didácticos se han incrementado en bastante porcentaje.

3.5 Asincronismo VNUMLGUI

El asincronismo es un concepto importante en la mayoría de los ámbitos de la computación, que también está presente en este proyecto.

La importancia de la concurrencia reside en poder ejecutar simultáneamente múltiples tareas interactivas, que pueden ser un conjunto de procesos o hilos reejecución creados por un único programa. Las tareas pueden ser ejecutadas en una sola unidad de proceso (multiprogramación), en varios computadores o en una red de computadores distribuidos.

El principal objeto de la programación concurrente es la interacción entre tareas, la secuencia correcta de interacciones o comunicaciones entre los procesos y el acceso coordinado a los recursos compartidos, son las ideas claves de la concurrencia, y aquí es donde nos basamos para levar a cabo ciertas operaciones implementadas en el proyecto que se basan en este concepto que mas adelante se detallarán.

Los mensajes proporcionan una solución al problema de la concurrencia de procesos que integra la sincronización y la comunicación entre ellos y resulta adecuado tanto para sistemas centralizados como distribuidos. Ésto hace que se incluyan en prácticamente todos los sistemas operativos modernos y que en muchos de ellos se utilicen como base para todas las comunicaciones del sistema, tanto dentro del computador como en la comunicación entre computadores. Las diferencias en los modelos usados para la sincronización de los procesos se debe a las distintas formas que puede adoptar la operación de envío del mensaje. En la ejecución asíncrona el proceso que envía un mensaje sigue su ejecución sin preocuparse de si el mensaje se recibe o no.

La motivación que nos llevó a indagar sobre este concepto fue la gran cantidad de tiempo que consumían algunas tareas en el simulador, y que mientras éstas se realizaban, el programa se quedaba bloqueado a la espera, de modo que es investigó una posible solución.

En el modelo asíncrono el sistema operativo se encarga de recoger el mensaje del emisor y almacenarlo en espera de que una operación de recibir lo recoja. Normalmente este tipo de comunicación tiene limitado la cantidad de memoria que puede utilizar una pareja en comunicación directa o un buzón en comunicación indirecta, para evitar así que un uso descontrolado pudiera agotar la cantidad de almacenamiento temporal del sistema. Esto es, la forma en que VNUMLGUI trata los procesos es similar a la forma en que lo hace windows, se trata de poner cada petición de usuario, los procesos que se van generando, en una cola, procesando siempre el más antigüo, el problema es que hasta que no se termina de ejecutar uno no se empieza a procesar el siguiente, de ahí la necesidad de la computación paralela

De este modo el asincronismo se planteó como solución perfecta. El siguiente reto era conseguir un método para impleméntalo, en esta línea encontramos Glib::MainLoop que, en términos generales, proporciona un tipo de bucle que vigila el procesamiento de cada evento y activa las acciones apropiadas en cada caso. Glib tiene además unos cuantos dispositivos para procesar eventos como son: Glib::Timeout, Glib::Idle, y io watch (Glib::IO->add_watch);está última es la que nos va a servir para nuestro propósito:

El principal objetivo de la función Gtk::io_add_watch es manejar evento io sin bloqueo, esta es su sintáxis:

io_id Gtk::io_add_watch($stream, $conditions, $callback)

- $stream: es el fichero que va a ser analizado.
- El parámetro $conditions es el encargado de lanzar la llamada al evento necesario.Puede ser de los siguientes tipos:
  - Gtk::IO_IN: stream está listo para leer
  - Gtk::IO_OUT: stream está listo para escribir.
  - Gtk::IO_PRI: Este es el canal con más prioridad para stream
  - Gtk::IO_ERR: stream está en estado de error,
  - Gtk::IO_HUP: stream ha sido desconectado (sighup)

Si la función llamada por Gtk::io_add_watch, es decir $callback, devuelve true, significa que gtk continuará el proceso, mientras que si devuelve false parará.

3.5.1 Procesamiento de colas de trabajo

Ésto ha sido aplicado a VNUMLGUI, mejorando el tiempo de respuesta en muchos estados, como parar la simulación en los que el programa se quedaba bloqueado.

Observando la ejecución del programa se vió que los puntos en los que VNUMLGUI se quedaba bloqueado eran aquellos en los que había un bucle de tipo while, en los cuales el programa no podía continuar hasta que se terminara la ejecución dentro del mismo, esto era lo que provocaba los excesivos tiempos de espera en el programa, que tenía como consecuencia más evidente un no refresco de la pantalla ya que el programa se bloqueaba.

Esto se materializó en la eliminación de los bucles problemáticos, siendo sustituidos por la función Glib::IO->add_watch, anteriormente explicada. El siguiente ejemplo muestra como ha sido la dinámica seguida, en términos generales:

```
Glib::IO->add_watch ( fileno(STATUS), ['in', 'hup'], sub {
        my ($fileno, $condition) = @_;
        if ($condition eq 'hup') {….(1) ….}
 …(2)…}
```

Con esto conseguimos leer el fichero STATUS hasta que lleguen evento in o hub, es decir, que el fichero tenga datos para leer (in) o haya habido una desconexión o error (hub), en este caso el código a ejecutar sería (1), mientras que (2) se ejecutaría en caso de no entrontrar errores.

4. SIMULACIÓN DE UN ESCENARIO COMPLEJO: COMPLU6IX

La motivación inicial de este proyecto fue la de proporcionar un entorno de simulación adecuado para el proyecto Complu6IX I :"transición del campus UCM a IPv6" en el que hipotéticamente se inyecta IPv6 en dual snack con IPv4, estudiando su viabilidad, ventajas, inconvenientes .

Hace 10 años, IPv6 era considerado un experimento con el que se había practicado de modo simulado utilizando el mismo IPv4 como vector de transporte. Hoy en día la conectividad IPv6 nativa está disponible a lo largo de diversos puntos en Internet, incluyendo áreas como Norte América, Europa, y Asia. En Europa la red GEANT ha ofrecido una red de IPv6 dual stack desde 2003, y la mayoría de Redes Nacionales para la Educación e Investigación (Research and Education Networks (NERNs)), han desplegado IPv6 como un servicio disponible más. Estos esfuerzos han sido impulsados por los resultados producidos por 6NET y Euro6IX, entre otros. En España, RedIris es el encargado de suministrar a las Universidades el acceso a Internet.

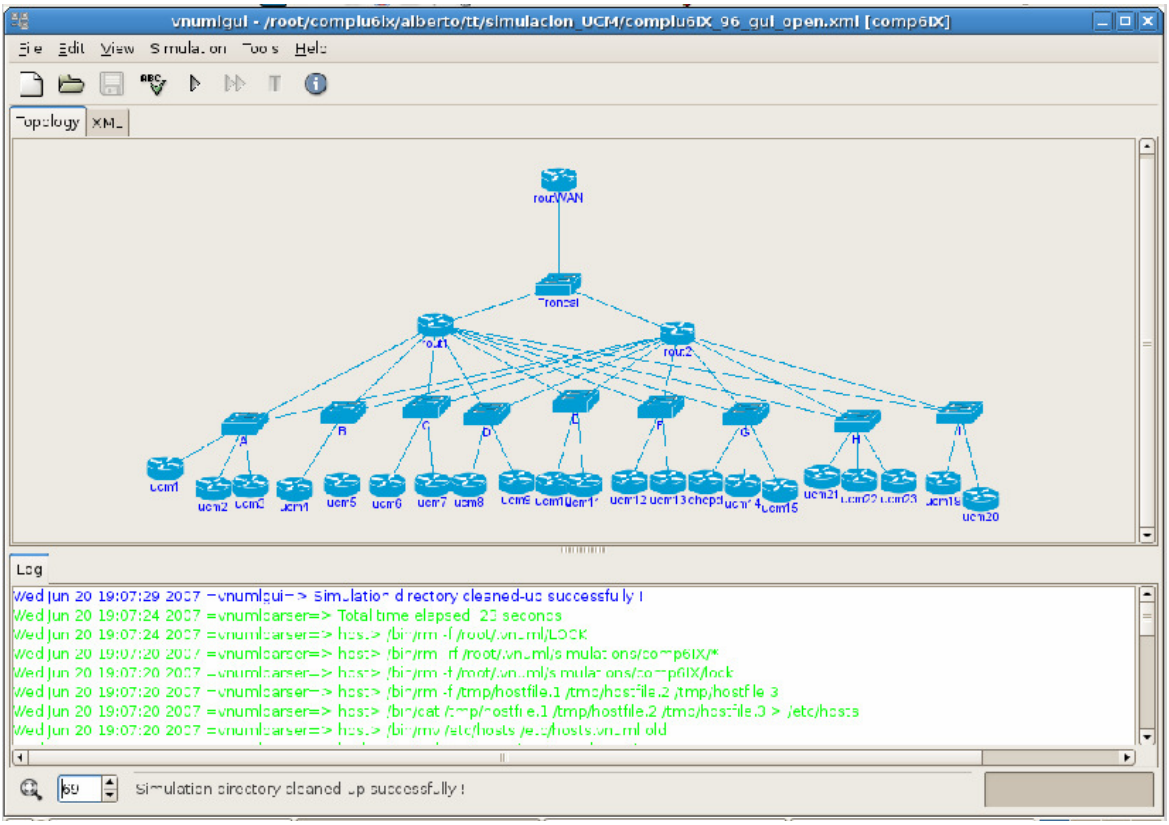
4.1 Breve descripción de la topología de la UCM.

La red de datos de la UCM interconecta y da acceso a internet de todos los centros de la Complutense, así como a otros centros relacionados con ella. Consta de dos grandes routers centrales, uno en el centro deprocesamiento de cálculo (CPD) y otro en el edificio de vicerrectorado, formando una configuración física en forma de doble estrella. El objetivo que se persigue con esta redundancia es que si el router primario deja de funcionar, todo el tráfico pasaría por el router secundario y se seguiría manteniendo la conectividad con Internet en toda la UCM. No todos los centros enlazan directamente con los routers centrales. Algunos centros enlazan con otros edificios que hacen de intermediario entre ellos y los routers centrales.

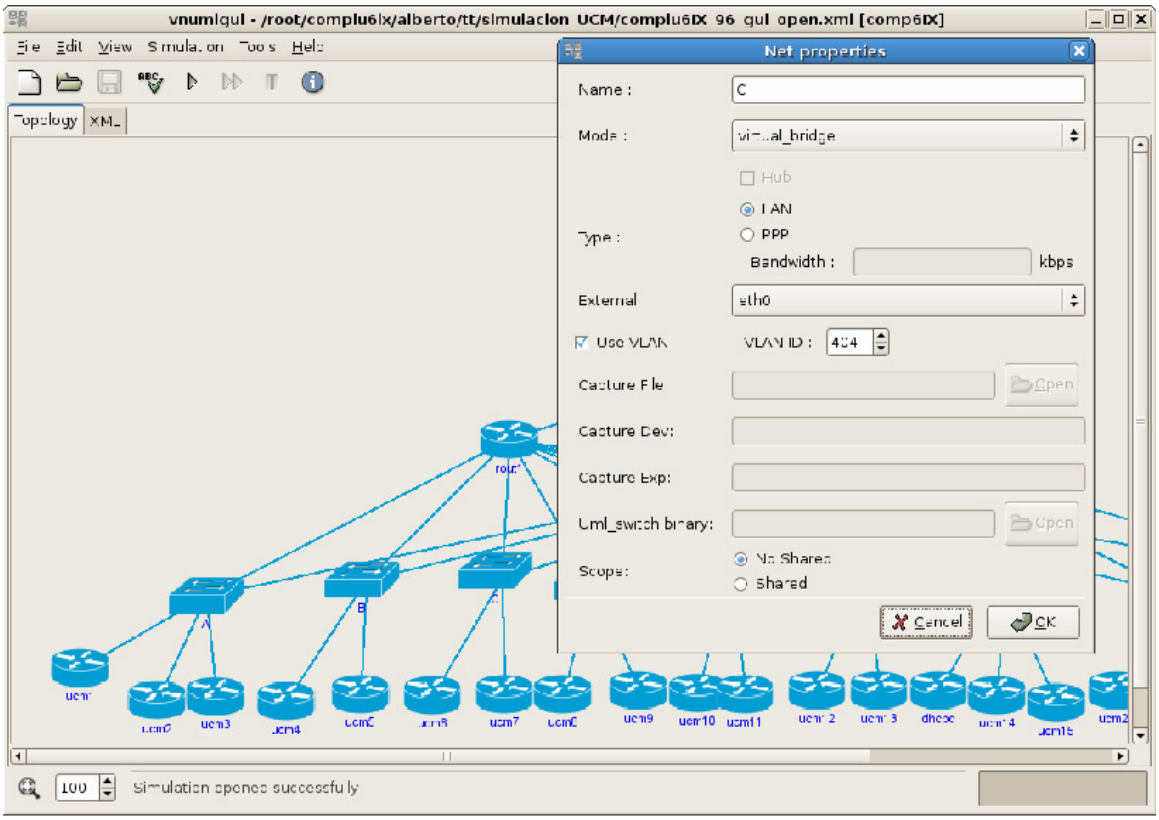
A continuación se van a mostrar algunas de las simulaciones que se han podido realizar mediante la colaboración entre ambos proyectos.

Se va a observar la disposición de los nodos que conforman la topología de la UCM. Los switches A, B, C, D… simbolizan una herramienta gráfica de interconexion, no existen en un entorno real, mientras que los nodos ucm1, ucm2….sí se corresponden con los terminales físicos reales. Net1 y Net2 representan los dos routers centrales encargados de mantener la doble estrella.

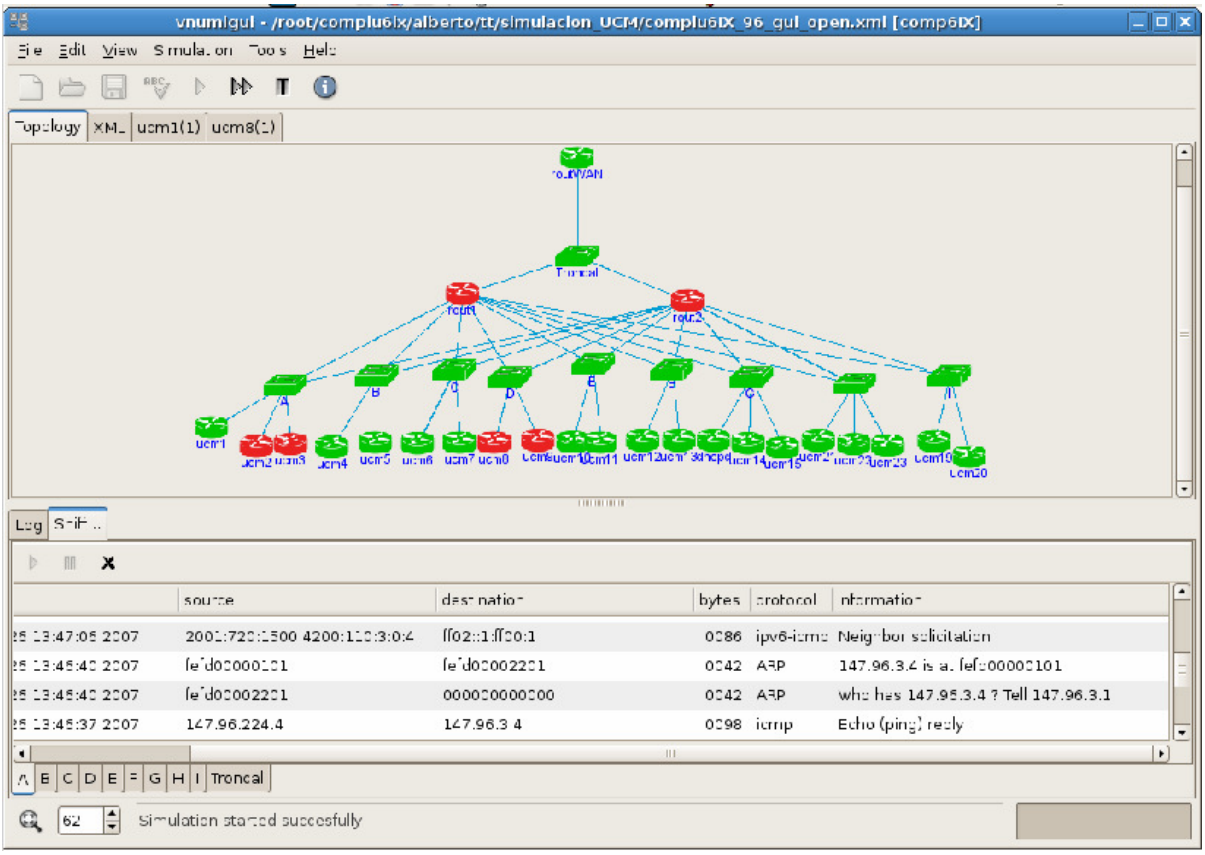
En esta primera captura se ve la disposición de la topologia de la UCM en el estado inicial antes de simular, en la parte inferior se observa la pestaña implementada para dar mas accesibilidad al usuario a la hora de visualizar los log.



Como hemos dicho para poder utilizar el analizador de protocolos hace falta que los switches sean modo virtual_bridge o uml_switch (siempre que se rellene el campo "capture_dev"):



Esta última captura se corresponde con la simulación en estado running. Los nodos rojos nos indican por dónde circula el tráfico en cada momento. Debido a que se iluminan sólo cuando reciben un número impar de paquetes, los colores de routers y switches cambian de color continuamente, cuando circula trafico sobre ellos. En la parte inferior está el sniffer donde se puede observar con detalle todo el tráfico de paquetes.



4.2 VLANS

Una 'VLAN' es un método para crear redes lógicas independientes dentro de una red física, de esta forma una red de computadores se comportarán como si estuviesen conectados al mismo cable, aunque pueden estar en realidad conectados físicamente a diferentes segmentos de una red de área local.

Los administradores de red configuran las VLANs mediante software en lugar de hardware, lo que las hace extremadamente flexibles. Una de las mayores ventajas de las VLANs surge cuando se translada físicamente una computadora a otra ubicación: puede permanecer en la misma VLAN sin necesidad de ninguna reconfiguración hardware.

Dentro de cada VLAN se practica un subnetting con fines meramente administrativos, expandiendo la red. El estudio de este tipo de redes, ha sido clave para la transición a ipv6, estudiando especialmente el subnetting implementado. Asi, cuando se quiere pasar de una subred a otra que esta en la misma VLAN, no es necesario el paso por el router del CPD.

5. CONCLUSIONES

- Independencia del formato del DTD

Como ya se ha explicado XML es el lenguaje base de VNUML, para la descripción de los escenarios de simulación, y la DTD es un archivo que encierra una definición formal de la estructura de etiquetas del documento y, a la vez, especifica la estructura lógica de cada documento, de forma que el fichero de VNUML es procesado mediante un parser validador basado en DTD, construyendo en la memoria del programa un árbol DOM (Dynamic Object Model), el programa se interrumpe si la entrada no se corresponde al modelo de DTD, esto es, el archivo VNUML no es formalmente correcto. Con cada versión del DTD, se tiene que programar explicitamente el generador de código de XML, la proyección del árbol sintáctico al XML, y múltiples líneas de código que suponen un gasto enorme de tiempo y esfuerzo para los programadores.

Se espera que a medida que VNUML aumente de versión, la actualización de VNUMLGUI sea cada vez más sencilla, evitando que la aplicación quede obsoleta y desfasada. Ya que VNUML sigue un ritmo de actualizaciones bastante acelerado, el mantenimiento de VNUMLGUI es complicado. Una posibilidad para reducir la complejidad y el esfuerzo que conlleva la actualización del software es la independencia del DTD. Esta es la clave para que el mantenimiento de VNUMLGUI se centre únicamente en el aumento de nuevas funcionalidades y con ello convertirlo en un software potente y con un amplio repertorio de posibilidades. La independencia del DTD formaría parte del trabajo del equipo de VNUML. Sería necesario incluir en el paquete VNUML un módulo en el que se realizarán las operaciones que involucran a la DTD, y desde VNUMLGUI bastaría con invocarlo.

-Modularidad del proyecto

La modularidad es la característica de los programas que hace que los componentes del mismo sean lo mas independientes posible, entre sí.

Esto aporta multitud de ventajas:

- Cada módulo se puede crear aisladamente y varios programadores pueden trabajar simultáneamente.

- Permite que se modifique un módulo sin que afecte a los demás, lo que implica que un error esté mucho más localizado y se pueda aislar fácilmente.

- El uso de módulos facilita la comprensión de la lógica subyacente para el programador y el usuario.

- El mantenimiento y la modificación de la programación se facilitan.

Entre las principales ventajas que ofrece perl está que es un lenguaje de propósito general, interpretado y ofrece un modelo orientado a objetos, que incluye referencias, paquetes y una ejecución de métodos basada en clases y la introducción de variables de ámbito léxico, que hace más fácil escribir código robusto (junto con el pragma strict). Una característica principal introducida en Perl fue la habilidad de empaquetar código reutilizable como módulos.

Pese a esta gran ventaja que ofrece Perl en este proyecto es desaprovechada, de forma que el programa script VNUMLGUI ocupa mas de 12.000 líneas, lo que le hace inmanejable, de forma que una de las posibles mejoras sería descomponer cada "package" en un fichero distinto, que ayude a manejar partes mas pequeñas y obtengamos todas las ventajas anteriormente detalladas que ofrece la modularidad y concretamente la modularidad en Perl.

-Integración con el VNUML. Diseño

Actualmente lo único que hace VNUMLGUI es invocar una orden de shell el sistema VNUMLPARSER. Si se consiguiera hacer que los paquetes de VNUMLGUI heredarán de las clases de VNUMLPARSER, automáticamente podrían invocarlos en el mismo espacio de memoria, sin tener que abrir costosos mecanismos como los "pipes".

APÉNDICE

A.1 Manual de instalación de VNUMLGUI

A.1.1 Prerrequisitos

Para que VNUMLGUI funcione y ejecute sus simulaciones es necesario tener instalado el paquete VNUML. Muchos de los módulos necesarios vienen ya instalados en la mayoría de las distribuciones Linux.

En la mayoría de casos, se necesita el kit 'developer' para acceder a las librerías necesarias. Un kit 'developer' es diferente de una distribución normal binaria. Generalmente, el paquete 'developer' incluye los binarios compilados para el sistema operativo para el que se va a trabajar. Si fuese posible, se deberá usar la última versión de los paquetes.

A.1.1.1 Librerías necesarias para VNUML.

Antes de instalar todos los módulos necesarios para VNUMLGUI, hay que centrarse en los que se necesitan para la instalación del paquete VNUML.

Para la correcta instalación de VNUML es necesario tener instalados los siguientes paquetes:

- Error
- Exception::Class
- XML::DOM,
- XML::Checker
- Estos módulos normalmente requieren la instalación de los siguientes: XML::RegExp, XML::Parser, XML::Parser::PerlSAX
- NetAddr::IP
- Net::Pcap
- IO::Socket
- Term::ReadKey
- Net::IPv6Addr
- Este módulo normalmente requieren la instalación de los siguientes: Math::Base85, Net::IPv4Addr

- File::Glob

Todos estos módulos se pueden instalar con ayuda del repositorio CPAN.

Una vez que este proceso está completado, se puede recurrir a la instalación de VNUML, que será satisfactoria si se instalaron todos los paquetes correctamente.

A.1.1.2 Librerías necesarias para VNUMLGUI.

Gtk2-perl

El paquete Gtk2-perl se requiere para el interfaz. Concretamente, se necesitan los módulos siguentes:

- Gtk2 1.061+ (http://gtk2-perl.sourceforge.net/)
- Gtk2::GladeXML 1.001+ (http://gtk2-perl.sourceforge.net/)
- Gtk2::SimpleList 0.15+ (http://search.cpan.org/~tsch/Gtk2-1.082/)

Gnome2-perl

Gnome2 perl contiene elementos avanzados para la implementación de Interfaces de Usuario y de los que VNUMLGUI hace uso, por ejemplo, el Canvas para el diseño de redes, VFS para la importación de los layout, Vte para los terminales virtuales empotrados.

- Gnome2 1.020+ (http://gtk2-perl.sourceforge.net/)
- Gnome2::Canvas 1.002+ (http://gtk2-perl.sourceforge.net/)
- Gnome2::VFS 1.003+ (http://gtk2-perl.sourceforge.net/)
- Gnome2::Vte 0.04+ (http://gtk2-perl.sourceforge.net/)

Módulos misceláneos

Locale::gettext 1.01+.

Durante el desarrollo de las nuevas funcionalidades incluídas en VNUMLGUI, fue necesaria la importación de dos nuevas librerías, tales como:

- NetPacket-0.04
- Libsocket6-perl

A.1.2 Instalación

El software completo puede ser descargado de la página de sourceforge (https://sourceforge.net/projects/vnumlgui/)

La instalación de vnumlgui es bastante simple y directa.

```
# tar xvzf vnumlgui-0.7.tar.gz
# cd vnumlgui-0.7
# make install
Changing vnumlgui prefix to /usr/local...
perl -ne 's!\@PREFIX\@!/usr/local!g; s!\@LIBDIR\@!/usr/local/lib/vnumlgui!g; print' < src
Creating doc...
Building tranlations...en...fr...
Creating directories under //usr/local...
Copying share files to //usr/local/share...
Chowning files in //usr/local/share/vnumlgui...
Installing binary (//usr/local/bin/vnumlgui) and man pages...
Installing translations (en, fr)...
```

El comando make instala VNUMLGUI por defecto bajo el directorio /usr/local/. Éste se puede cambiar especificando un PREFIX diferente:

```
# tar xvzf vnumlgui-0.7.tar.gz
# cd vnumlgui-0.7
# make install PREFIX=/usr
Changing vnumlgui prefix to /usr...
perl -ne 's!\@PREFIX\@!/usr!g; s!\@LIBDIR\@!/usr/lib/vnumlgui!g; print' <src/vnumlgui >
Creating doc...
Building tranlations...en...fr...
Creating directories under //usr/...
Copying share files to //usr/share...
Chowning files in //usr/share/vnumlgui...
Installing binary (//usr/bin/vnumlgui) and man pages...
Installing translations (en, fr)...
```

El proceso de desinstalación también es sencillo:

```
# make uninstall
rm -rf /usr/local/bin/vnumlgui /usr/local/share/man/man1/vnumlgui.1.gz
/usr/local/share/vnumlgui/vnumlgui.glade
```

A.2 Manual de uso de VNUMLGUI

A.2.1 Introducción:

¿Qué es VNUMLGUI?

VNUMLGUI es una herramienta gráfica para ayudar al usuario a crear, corregir y ejecutar simulaciones de VNUML, sin tener crear archivos XML de simulación.

Lo más destacado de todo es que VNUMLGUI es un editor de la topología de las redes con capacidades gráficas. El usuario puede dibujar virtualmente su red, situando los routers (VMs) y los switches (Nets). Enlazándolas unas con otros mediante cables, el usuario puede crear cualquier topología que necesite y ver el comportamiento de sus experimentos de la red.

Con VNUMLGUI, se pueden crear y ejecutar simulaciones simples en menos de 1 minuto. VNUMLGUI oculta la parte compleja de VNUML y de XML.

A.2.2 Ejecución de VNUMLGUI

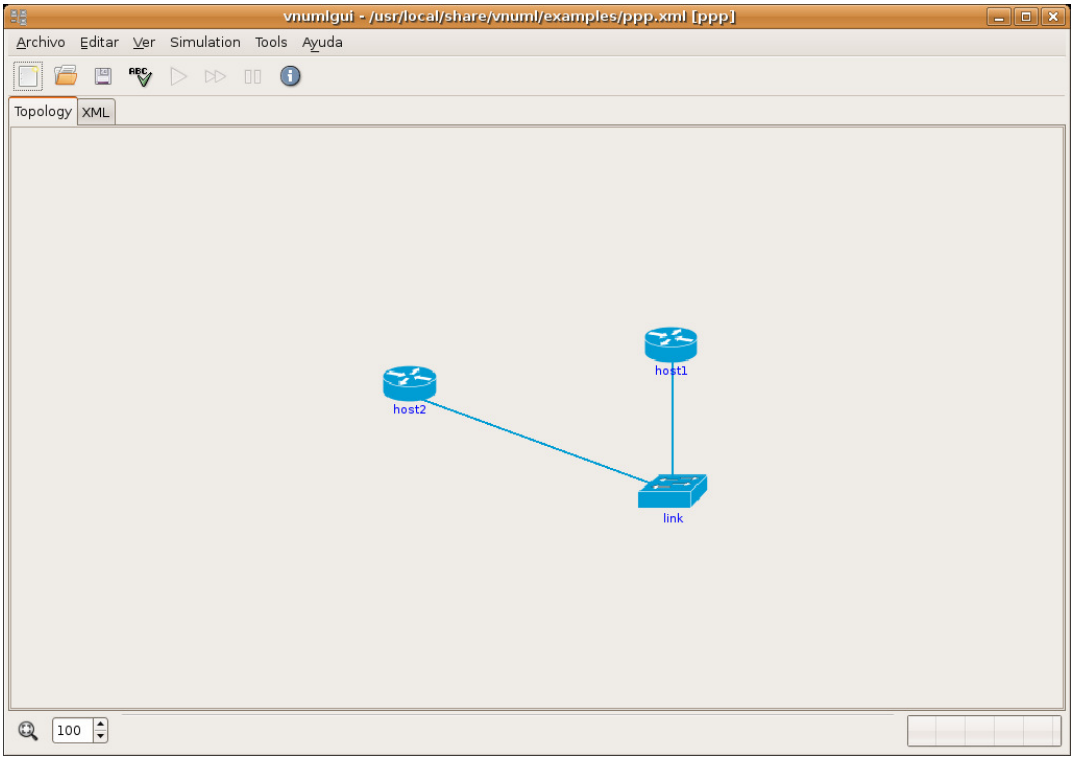
VNUMLGUI tiene únicamente una opción de línea de comando: -g. Se utilizará cuando se necesite ejecutar VNUMLGUI en modo depuración, y así poder ver información en la consola.

Se recomienda ejecutar VNUMLGUI la primera vez sin ninguna opción y configurar los parámetros necesarios para que aparezcan por defecto en las posteriores simulaciones y hacer así su utilización más sencilla.

La primera vez que se abre VNUMLGUI, éste crea un archivo ~/.vnumlgui/config en el que se guarda la configuración de los parámetros de VNUMLGUI.

Ventana principal de la aplicación.

La ventana principal del VNUMLGUI se divide en varias partes: la barra de los menús, la barra de herramientas, un notebook con varias pestañas, y la barra de estado.

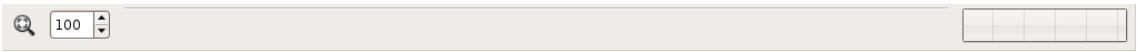


La barra de herramientas.

Esta barra proporciona acceso directo a las funciones más usadas en VNUMLGUI. Los iconos que se muestran se describen a continuación por orden, empezando por la izquierda.

- New: crea una nueva simulación.
- Open: abre simulaciones existentes.
- Save: guarda la simulación.
- Check: analiza la sintaxis XML del fichero VNUML.
- Start (build) simulation: ejecuta la actual simulación con vnumlparser.pl -t.
- Exec scenario: Abre la ventana Exec, la cual permite ejecutar escenarios.
- Stop simulation: Para la simulación que se está ejecutando en ese momento con vnumlparser.pl -d.
- Information: Abre un panel inferior en la ventana principal de la aplicación. Muestra los Logs de la simulación y las capturas de paquetes del analizador de protocolos

La barra de estado.



Esta barra contiene los siguientes elementos de izquierda a derecha:

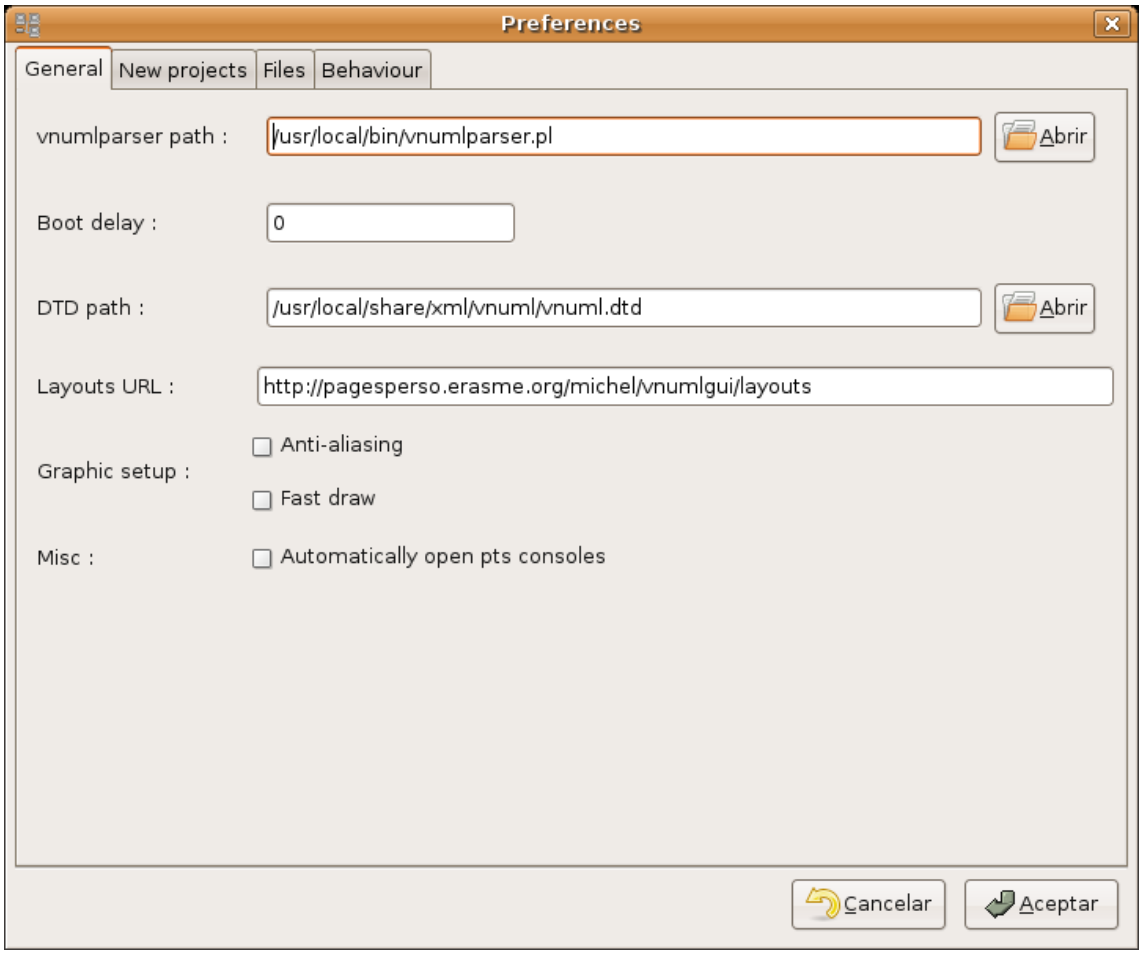
- Un botón “zoom to fit”
- Un indicador del factor de zoom. Las flechas pueden usarse para incrementar/decrementar el nivel de zoom. También se puede escribir en la entrada el valor del factor para fijar el nivel de zoom que se desea.
- Un indicador textual de estado el cual nos permite saber qué está ocurriendo.
- Un indicador de progreso para largas operaciones.

A.2.3 Configurar la aplicación

Las preferencias de la aplicación se configuran a través de la entrada de menú Edit -> Preferences. Aquellas opciones que se configuren se guardarán en el fichero ~/.vunmlgui/config.

Las opciones de esta ventana se dividen en cuatro grupos: preferencias generales de uso, valores por defecto a utilizar en nuevos proyectos, rutas de los ficheros relacionados con kernel/filesystem/ssh, y ordenes apara indicar qué hacer con los elementos desconocidos que se encuentran al abrir nuevas simulaciones.

A.2.3.1 Preferencias Generales

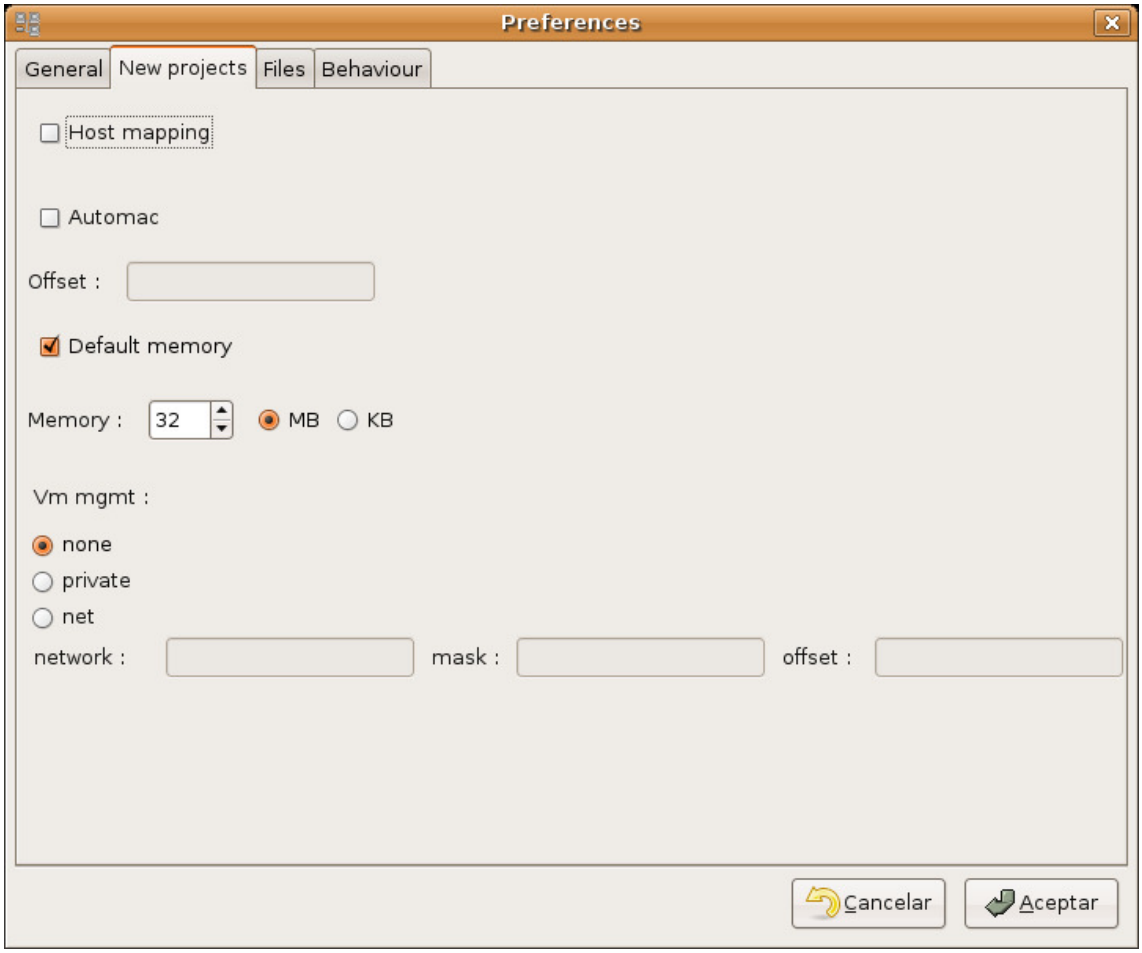


Esta ventana contiene las entradas para la configuración general de la aplicación. La siguiente tabla resume los diferentes valores y asocia un "key name" usado para guardar cada valor en el archivo de configuración (~/.vnumlgui/config).

Entrada	***Nombre Clave***	***Descripción***
vnumlparser path	vnuml.vnumlparser	Ruta absoluta de vnumlparser.pl
boot delay	vnuml.boot_delay	Parámetro de retraso de arranque que se pasa a vnumlparser.pl con la opcion -w.
DTD path	vnuml.dtd	Ruta absoluta del fichero vnuml DTD.
VNUML Version	vnuml.version	Versión de vnuml utilizada. Actualmente soporta la 1.7.0-

		1.
Configuración gráfica - Anti-aliasing	ui.anti_aliasing	El uso de anti-aliasing hará que la apariencia de los diagramas sea más pulida, pero ello retrasará el refresco de la pantalla mucho. Para simulaciones muy grandes es preferible deshabilitar el anti-aliasing. Si se cambia la configuración, es necesario reiniciar la aplicación para que los cambios tengan efecto.
Configuración gráfica - Fast-draw	ui.fast_draw	Si esta opción se selecciona, el movimiento de los enlaces cuando se mueve algún elemento del diagrama (Switches, Virtual Machines y Host) no se hará en tiempo real. Los enlaces serán redibujados cuando el movimiento haya terminado. Como consecuencia de ésto, el refresco de la pantalla será más rápido en simulaciones grandes, pero tiene el incinveniente de que no se observa el efecto en los enlaces cuando se mueven. La configuración tendrá efecto sin necesidad de reiniciar la aplicación.
Misc - Automatically open pts consoles	ui.auto_open_pts	Si está seleccionada esta opción, vnumlgui abrirá automáticamente las consolas pts cuando las máquinas virtuales las usen para el arranque.
Layouts URL	ui.layouts_url	URL usada para buscar layouts de las simulaciones en internet. Si se quiere ajustar su propio repositorio, el fichero layout es el mismo que el vnuml layout en local (~/.vnumlgui/layouts).

A.2.3.2 Nuevos Proyectos.

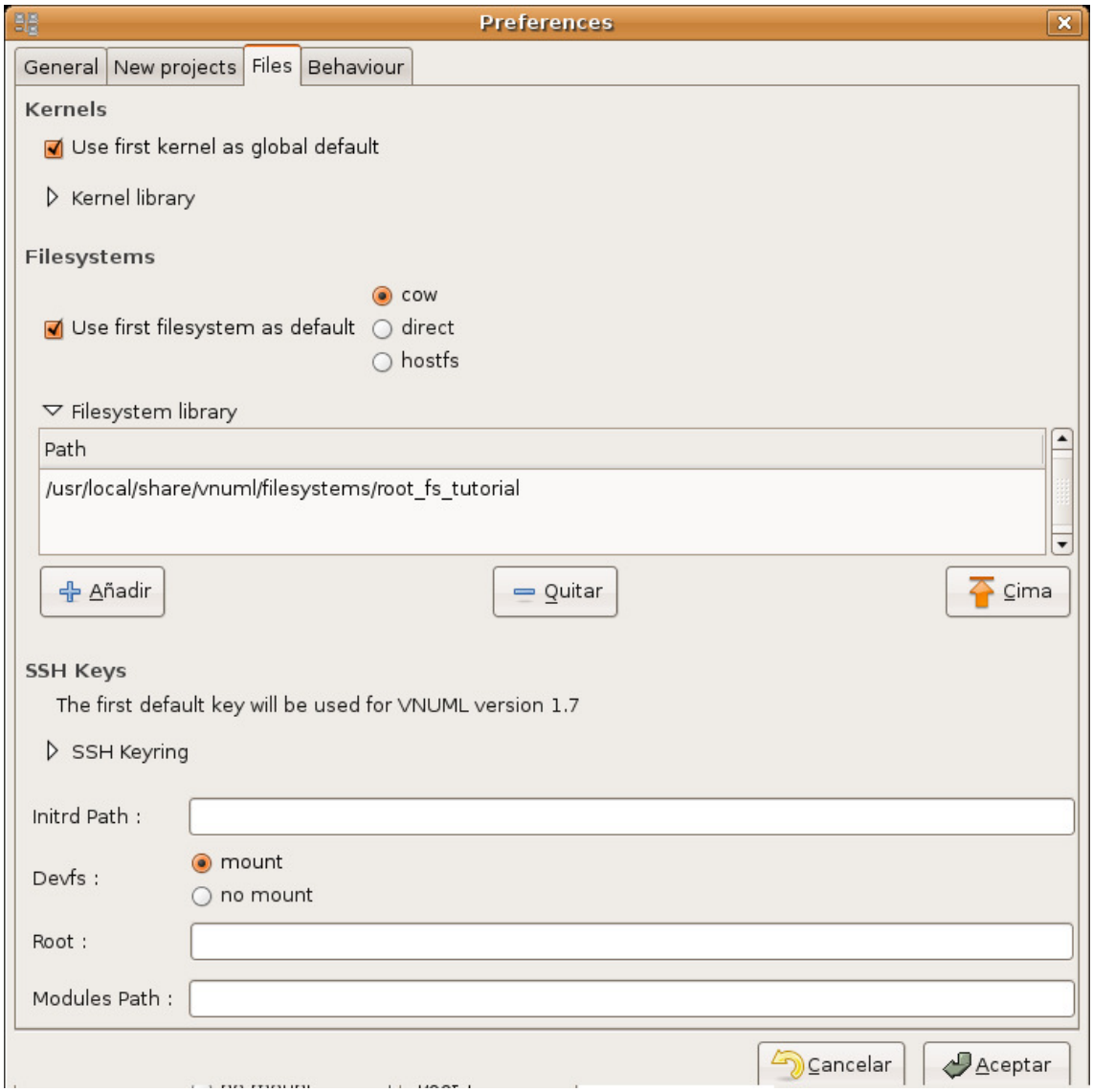


Las siguientes etiquetas son parte de vnuml DTD y su descripción se encuentra en la página oficial de vnuml (http://www.dit.upm.es/vnumlwiki/index.php/Reference).

Entrada	***Nombre Clave***	***Descripción***
Host mapping	vnuml.host_mapping	Para fijar host mapping o no en las nuevas simulaciones.
Automac	vnuml.automac	Para usar automac o no en las nuevas simulaciones.
Automac - Offset	vnuml.automac_offset	Valor opcional que se habilita cuando 'Automac' ha sido seleccionado.
IP Offset	vnuml.ip_offset = 0	Para usar ip offset o no en las nuevas simulaciones.

IP Offset - Prefix	vnuml.ip_offset_prefix	Valor opcional que se habilita cuando 'IP Offset' ha sido seleccionado.
IP Offset - Offset	vnuml.ip_offset > 0	Valor opcional que se habilita cuando 'IP Offset' ha sido seleccionado.
Default memory	vnuml.memory > 0	Para fijar la cantidad de memoria de cada maquina virtual.
Default memory - Memory	vnuml.memory	Para fijar la cantidad de memoria de cada maquina virtual.

A.2.3.3 Preferencias de ficheros.



Esta etiqueta se encarga de los ficheros de los kernels, de los filesystems y de las claves ssh, y almacena las rutas para no tener que escribirlas cada vez que se construye una simulación. Se pueden agregar tantas rutas como se quiera en cada categoría pulsando sobre el botón “Add”. Las nuevas entradas se van agregando desde el final, pero si se desea utilizar un determinado fichero por defecto como global (en el caso de “Kernels” o “Filesystems”, hay que poner la ruta al principio de la lista con el botón “Top”. Cuando el elemento “Use first... as global default ” ha sido seleccionado, el primer fichero de la lista será incluido en la sección <global> de VNUML.

Los kernels, filesystems y ssh keys son almacenados respectivamente en vnuml.kernels, vnuml.filesystems and vnuml.keys en el archivo de configuración (~/.vnumlgui/config).

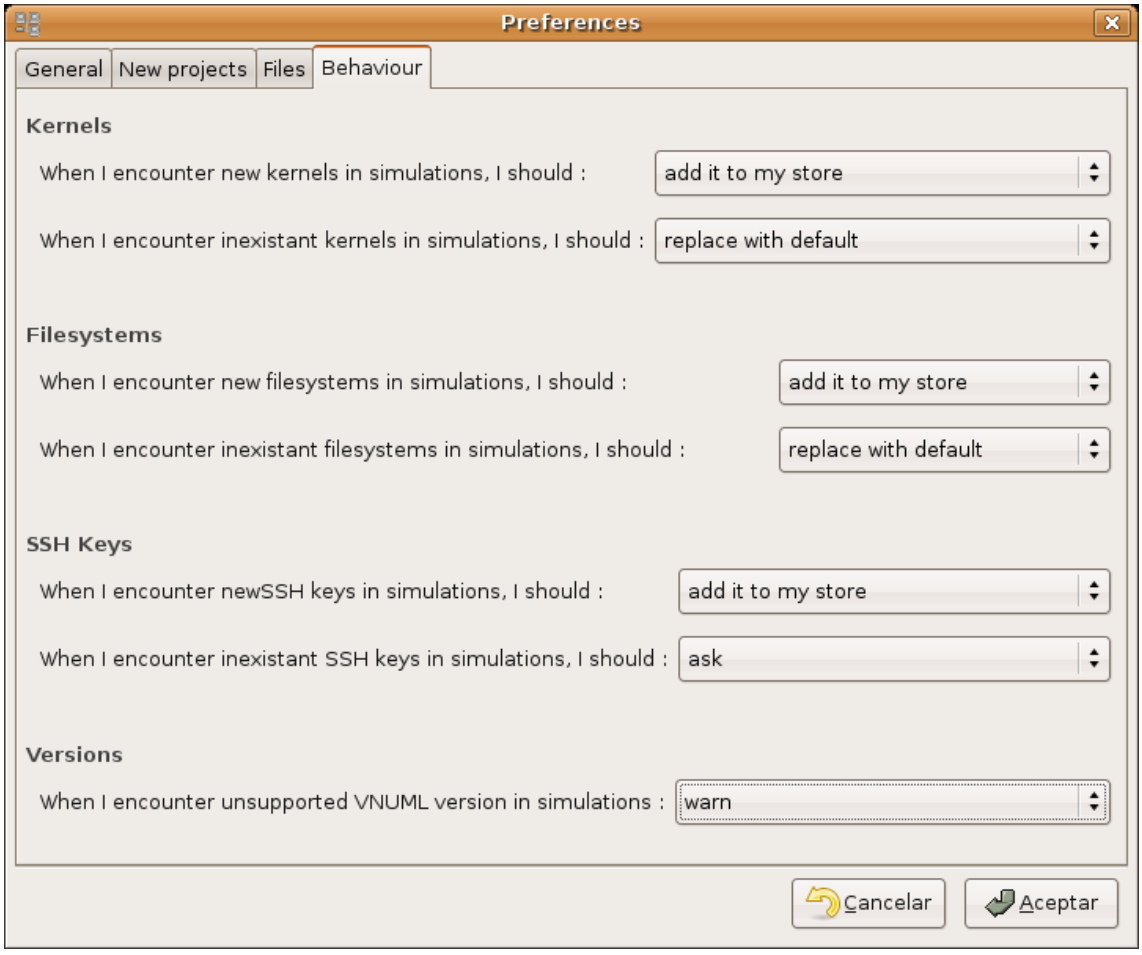
Para indicar que ’Use first... as global default’ ha sido seleccionado los valores vnuml.use_default_kernel and vnuml.use_default_filesystem son fijados a 1.

Además en vnuml.default_filesystem_type se fija el valor que se haya seleccionado entre “cow”, “direct” y “hostfs”.

Como características añadidas a los kernels existen los campos “Initrd Path”, “Devfs”, éste puede ser “mount” o “no mount”, “Root” y “Modules Path”.

Todos estos valores son almacenados en vnuml.kernel_initrd, vnuml.kernel_devfs, vnuml.kernel_root, vnuml.kernel_modules.

A.2.3.4 Preferencias en el comportamiento del arranque.



Cuando se abren nuevas simulaciones con VNUMLGUI, puede que la aplicación se encuentre en situaciones desconocidas o insostenibles. Por ejemplo, un kernel que no se encuentre en la lista “Kernel Library”, o una versión de DTD que no tiene soporte en VNUMLGUI y por tanto no se podrán traducir todos sus elementos, etc...

Los parámetros de esta pestaña permiten asignar un comportamiento por defecto a VNUMLGUI para que sepa cómo responder automáticamente ante estos eventos.

Al abrir simulaciones existentes, considerando las localizaciones de los ficheros, dos situaciones pueden presentarse:

- El fichero referenciado existe en el filesystem, pero no se encuentra en la librería de ficheros de VNUMLGUI.

- El fichero referenciado no existe en el filesystem.

Si la primera situación ocurre se pide a VNUMLGUI que actúe de alguna de las siguientes maneras:

- nothing: no se hará nada, y el valor se dejará tal cual. Esto significa que la simulación debería funcionar pero el fichero (kernel, filesystem o ssh key) no se añadirá a la librería de VNUMLGUI.

- add it to my store: el fichero referenciado será incluído a la librería de VNUMLGUI.

- ask: VNUMLGUI preguntará qué hacer (nada o incluirlo).

Cuando se presenta la segunda situación (el fichero no existe en el filesystem) se pide a VNUMLGUI que actúe de alguna de las siguientes maneras:

- nothing: no se hará nada, y el valor se dejará tal cual. En este caso la simulación no funcionará.

- replace with default: El fichero referenciado será reemplazado por el que esté seleccionado por defecto en la pestaña “Files”.

- warn: se comporta igual que ’nothing’, pero con la diferencia de que muestra un mensaje al usuario.

- ask: VNUMLGUI preguntará qué hacer (nada o reemplazarlo).

A.2.4 Simulaciones

A.2.4.1 Abrir nuevas simulaciones.

Una nueva simulación se puede abrir mediante el botón “Open” de la barra de herramientas o mediante el menú Archivo->Abrir.

En los dos casos aparece una ventana en la cual se puede seleccionar el archivo XML de la simulación que se quiera abrir.

A.2.4.2 Crear nuevas simulaciones.

Para crear una nueva simulación se puede hacer a través del botón “New” de la barra de herramientas o mediante el menú Arhivo->Nouveau.

En los dos casos aparece un cuadro de diálogo con una entrada de texto para escribir el nombre de la nueva simulación que se va a crear.

A.2.4.3 Ejecutar y parar simulaciones.

Un escenario que se haya creado o abierto mediante VNUMLGUI, se puede simular pulsando el botón "Build" o mediante el menú Simulation->Build.

Pero antes de hacer esto es recomendable seguir los siguientes pasos:

- Comprobar que la sintaxis de la simulación es correcta y está acorde con la versión de VNUML DTD que sigue VNUMLGUI. Si esto no se comprueba y hay errores, la simulación no llegará a ejecutarse.

- Borrar el directorio de la simulación. Esta acción borra cualquier instancia de la simulación en los directorios ~/.vnuml/simulations y ~/.vnuml/networks y el archivo ~/.vnuml/LOCK, mediante el botón "Clean up simulation directory" del menú "Simulation". Si a través de este botón, en los Logs se indica que no se pueden borrar los directorios, se deberá hacer esta acción a mano con permisos de root.

- Por último pulsar el botón "Build" para ejecutar la simulación.

Si se desea terminar la simulación, se puede parar mediante el botón "Release simulation" de la barra de herramientas o mediante el menú Simulation->Release.

Si la simulación no se puede detener de esta forma, existe como alternativa el botón "Forced Simulation" del menú Simulation.

A.2.4.4 Interactuar con simulaciones en ejecución.

A.2.4.4.1 Capturar el tráfico generado:

El tráfico generado solo se puede capturar cuando la simulación esta en ejecución. El Sniffer implementado consta de un analizador de protocolos, y de un "coloreador". Dependiendo de lo que se quiera será necesario hacer una cosa u otra. Las diferentes posibilidades se describen a continuación:

- Si se quiere capturar TODO el tráfico en tiempo real se debe pulsar "Ver->sniff all nets": de esta forma aparecerán en la parte de abajo una pestaña "Sniff", con varias subpestañas, una por red capturada. Además se colorearán los routers y switches por los que circulen los paquetes.

- Si se quiere capturar solo una red en especial: se pulsa botón derecho sobre el switch: "Sniff…" de esta forma se mostrará una pestaña en la parte inferior de la pantalla con la información recogida.

- Si se quiere "colorear" un solo router, entonces se pulsa botón derecho sobre el router elegido:"Highlight".

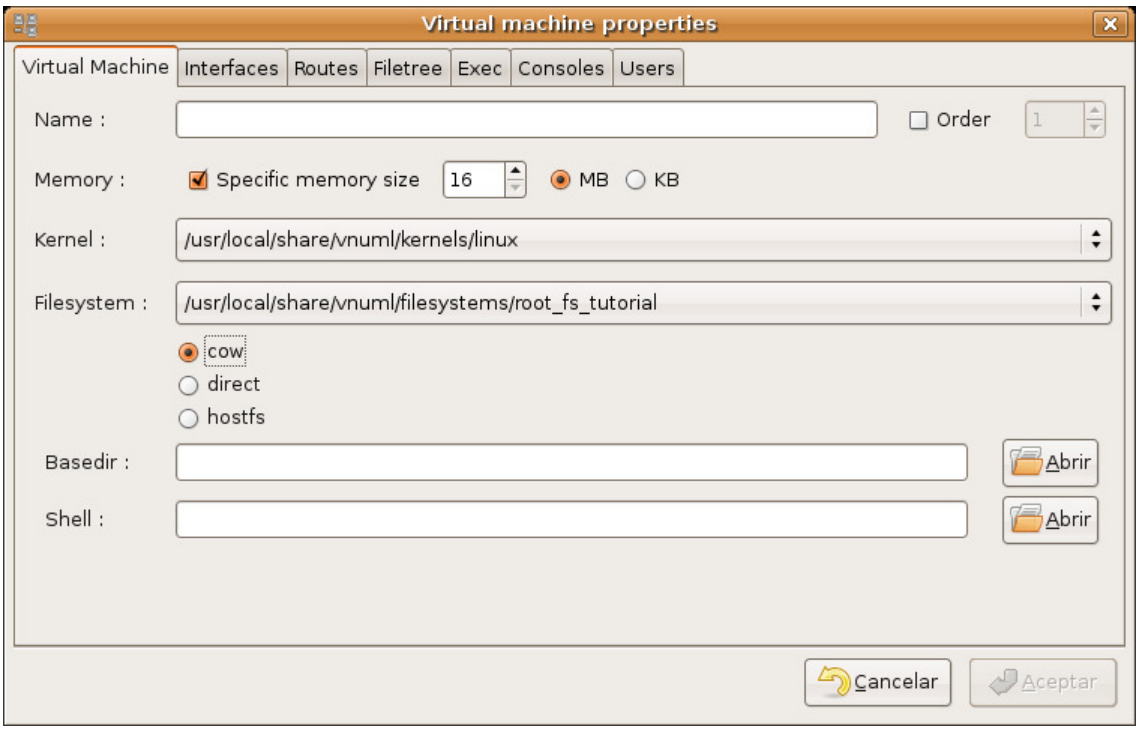
- Si se desea capturar los paquetes en un fichero, entonces se debe rellenar el campo “Capture file” de cada switch a capturar. Normalmente estos ficheros se almacenarán en tmp y tienen una extensión.cap:Ejemplo:/tmp/capture0.cap.

- Si se desea aplicar un filtro al fichero almacenado, se debe rellenar el campo “Capture Exp” de cada switch a capturar. Para ello se debe conocer la sintáxis del estándar de filtros.

A.2.5 Interfaz de usuario

A.2.5.1 Máquinas Virtuales.

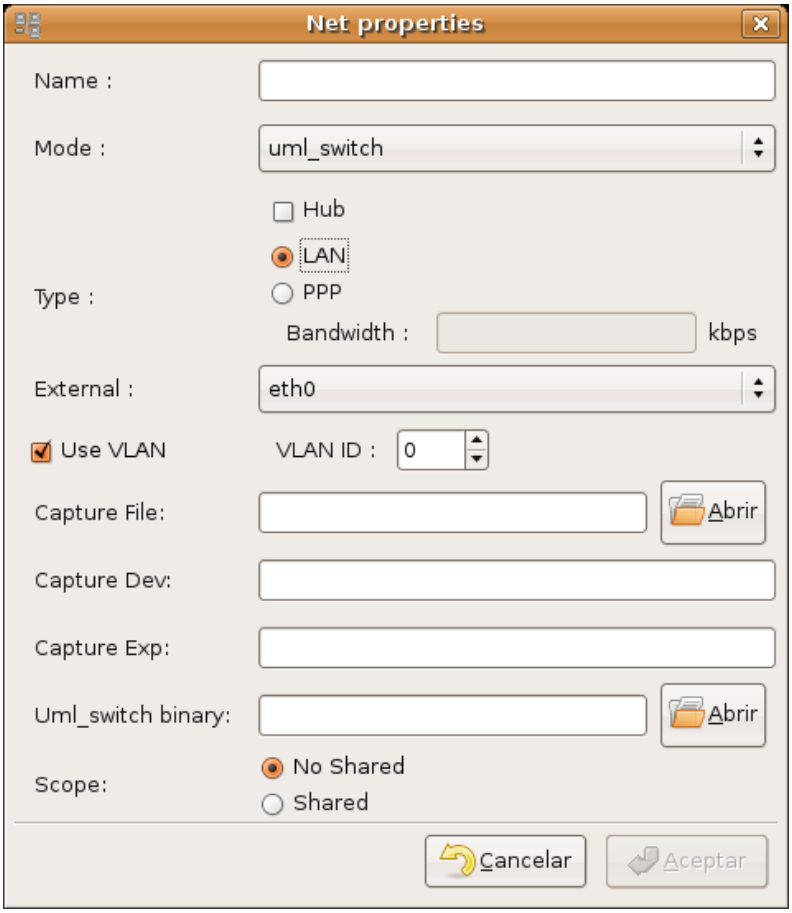
Para crear máquinas virtuales se pulsa con el botón derecho del ratón en cualquier sitio del panel “Topology”, en el menú desplegable que aparece se selecciona la opción “Add Virtual Machine”. Seguidamente se abre la ventana “Virtual machine properties”.



En esta ventana el único campo obligatorio es el nombre de la máquina virtual, “Name”. Cuando este campo de texto se rellena, se habilita el botón “Aceptar” de la ventana, mientras tanto permanecerá deshabilitado.

A.2.5.2 Redes.

Para crear redes se pulsa con el botón derecho del ratón en cualquier sitio del panel “Topology”, en el menú desplegable que aparece se selecciona la opción “Add Net”. Seguidamente se abre la ventana “Net properties”.

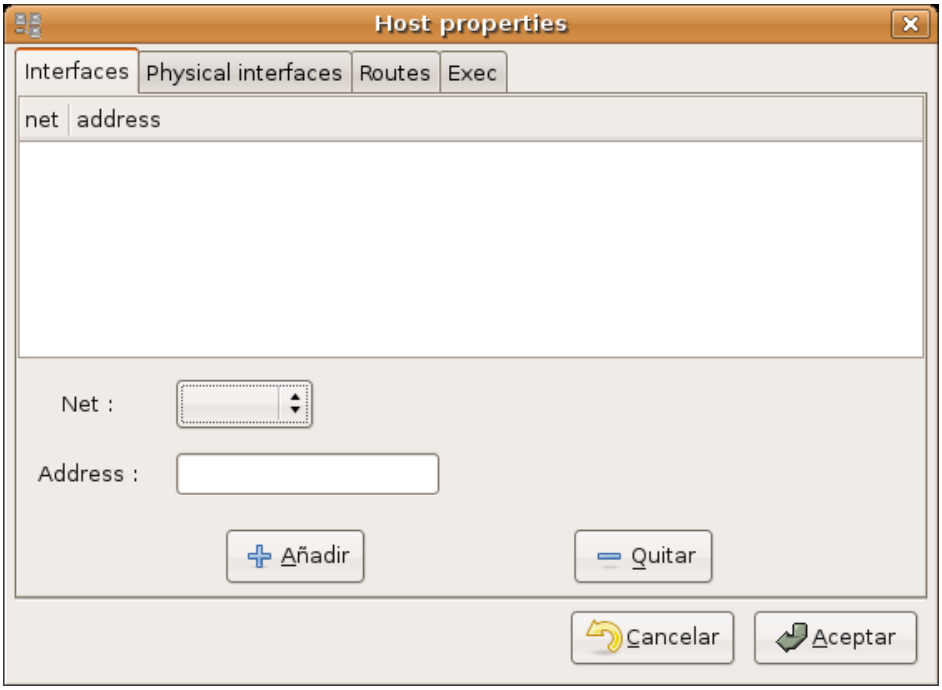


En esta ventana hay dos campos obligatorios, uno es el nombre de la red, “Name”, y otro es el Modo de actuación de la red, “Mode”.

Cuando el campo “Name” se rellena y el campo “Mode” toma algún valor, se habilita el botón “Aceptar” de la ventana, mientras tanto permanecerá deshabilitado.

A.2.5.3 Hosts.

Para crear hosts se pulsa con el botón derecho del ratón en cualquier sitio del panel “Topology”, en el menú desplegable que aparece se selecciona la opción “Add Host”. Seguidamente se abre la ventana “Host properties”.



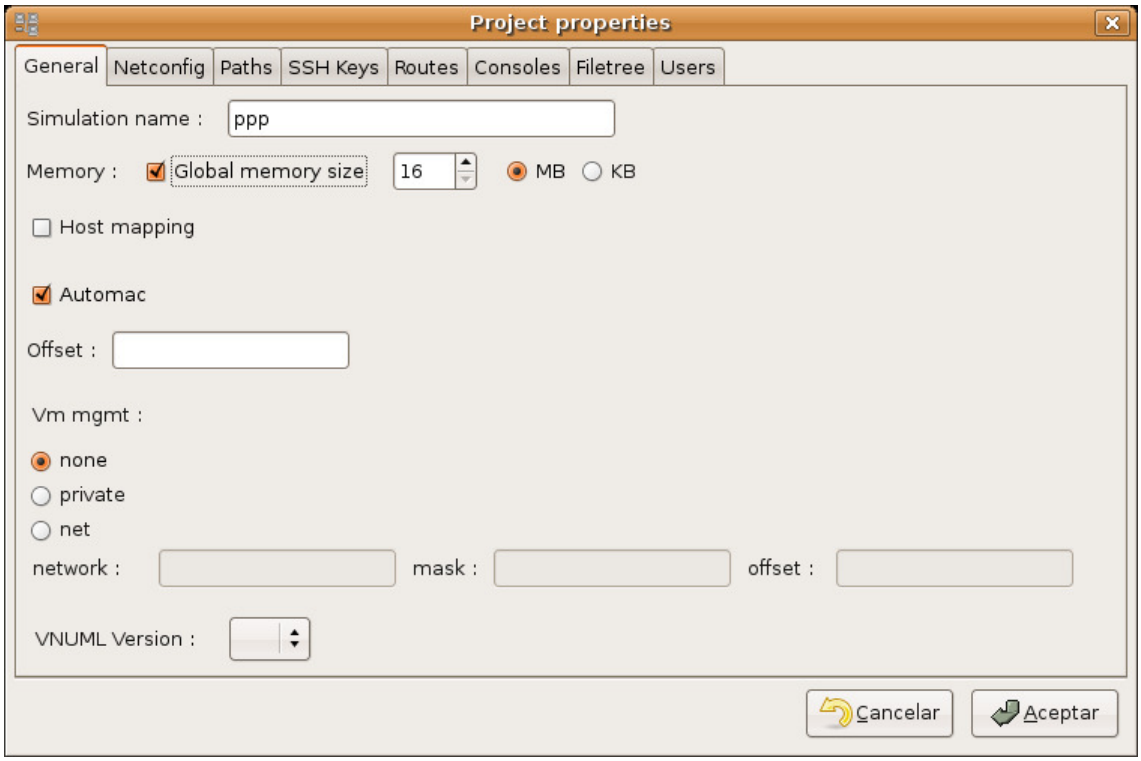
En esta ventana todos los campos son opcionales.

A.2.5.4 Preferencias.

En la sección “6.1.4 Configurar la aplicación” están descritas la mayoría de las características que conciernen a las preferencias de los proyectos.

A esta ventana se accede a través del menú Edit -> Preferencias.

A.2.5.5 Propiedades de los proyectos.



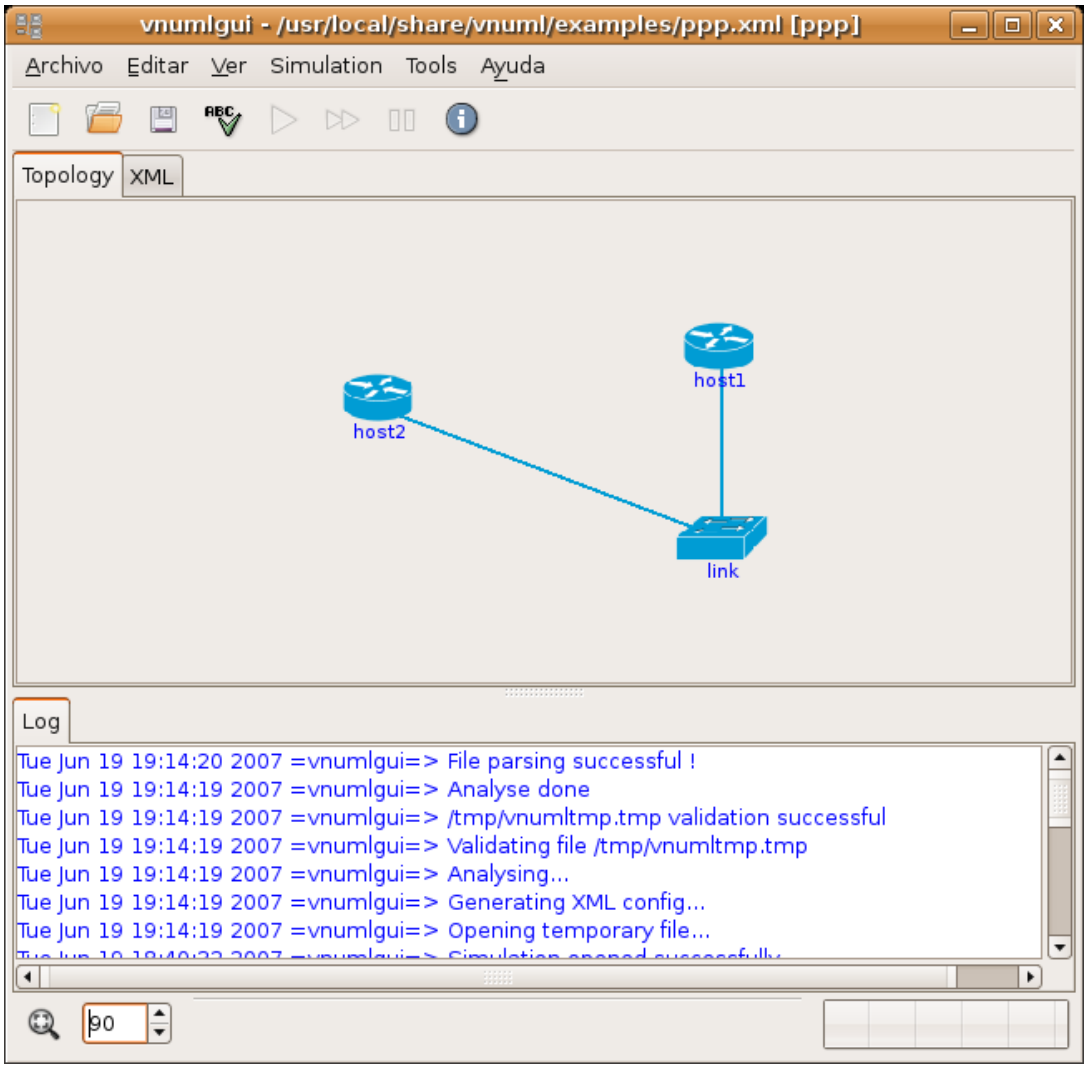
En esta ventana se configuran todas las características de los nuevos proyectos y que irán en la parte <global> del archivo XML que se genera.

Los datos que se introduzcan en esta ventana serán propios de cada proyecto que se crea o se abre con VNUMLGUI. Los valores introducidos no se conservan en posteriores proyectos, como ocurre en la ventana Preferencias, ya que no son guardados en ningún fichero externo de configuración, en cada nuevo proyecto que se abre o se crea se introducen los datos que se desean.

A.2.5.6 Logs

Las líneas de logs que una simulación genera, se pueden observar en VNUMLGUI mediante el botón de la barra de herramientas, o a través del menú Ver -> Logs.

De esta manera se abre un nuevo panel en la parte inferior de la aplicación, en la cual van apareciendo las líneas de logs.



A.2.5.7 Execute

Cuando una simulación se está ejecutando (después de haber pulsado el botón “Build”), si se desea ejecutar un escenario específico, que exista en el archivo XML de la simulación mediante la etiqueta <exec>, se debe pulsar sobre el botón “Execute scenario” de la barra de herramientas o a través del menú Simulation -> Execute scenario.

A.2.6 Accesos directos

Menú (Acceso directo)	*Contexto*	*Acción*
Doble click botón izquierdo del ratón	Sobre un elemento	Muestra la ventana de las propiedades del elemento (Máquina Virtual, Red, Host)
Ctrl- Doble click botón izquierdo del ratón	Sobre un elemento	Vuelca el hash objeto del elemento a STDOUT y muestra la ventana de las propiedades del elemento (Máquina Virtual, Red, Host.
Ver−→Zoom + (Ctrl-Rueda del ratón hacia delante)	Sobre panel central	Incrementa zoom
View−→Zoom - (Ctrl-Rueda del ratón hacia atrás)	Sobre panel central	Decrementa zoom
Ayuda+Debugging−→ Dump %CONFIG (F2)	Aplicación	Vuelca el hash %CONFIG a STDOUT. Se usa con propósito de depuración.
Ayuda+Debugging−→ Dump %STORE (F3)	Aplicación	Vuelca el hash % STORE a STDOUT. Se usa con propósito de depuración.
Ver−→Logs (F4)	Aplicación	Muestra el panel de Logs
Ver−→Logs (Doble click botón izquierdo del ratón)	Barra de estado	Muestra el panel de Logs
F5	Sobre panel central	Refresca la pantalla
Simulation −→Syntax check (F7)	Aplicación	Comprueba la sintaxis del fichero vnuml XML.
Simulation−→Build (Shift-F10)	Aplicación	Construye la topología de la simulaión.
Simulation−→Execute (Shift-F11)	Aplicación	Muestra la ventana Execute
Simulation−→Release (Shift-F12)	Aplicación	Detiene la simulación.
Simulation−→Forced release (Ctrl-Shift-F12)	Aplicación	Fuerza a que la simulación se detenga.